1988

The Development of a Compiler Design Course With Ada as The Implementation Language

Samuel L. Gulden
Department of EECS
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015

Darpa Grant #MDA972-92-J-1005

Final Report

The Development of a Compiler Design Course With Ada as The Implementation Language

Samuel L. Gulden

Department of EECS

Lehigh University

19 Memorial Drive West

Bethlehem, PA 18015

Darpa Grant #MDA972-92-J-1005

Final Report

1988

The Development of a Compiler Design Course With Ada as The Implementation Language

Samuel L. Gulden
Department of EECS
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015

Darpa Grant #MDA972-92-J-1005

Final Report

Table of Contents

Objectives

The purpose of this project was to introduce the language Ada into our compiler design curriculum as the development language. To this end our course which had been traditionally taught using Pascal was to be reworked so that the programming portion would now be done in Ada. In addition, while a subset of Pascal had been normally used as the target design compiler, the course was now to have a language similar to a subset of Ada as the target design compiler. This in fact was carried out and the ensuing report describes some aspects of the course development.

Initial Considerations

Ada had never been used as a programming language at Lehigh. Thus it was necessary to introduce Ada during the course. Fortunately the students in the class were all accomplished programmers familiar with Pascal. Thus it was possible to initially restrict the use of Ada to portions which most resembled Pascal and then introduce more concepts as we progressed. A more daunting task for the writer was to develop a syntax for the language which was to be the ultimate language to be compiled. The writer believes that LL(1) syntaxes are the most amenable to compiler development. Since no really good automated tool is availabie an LL(1) form of a subset of Ada which includes packages had to be developed. The LRM for Ada was not too useful in this regard. However by using some portions of Pascal and some additions a suitable syntax was developed. This syntax gave a language which strongly resembles a small but useful subset of Ada with some restrictions. Examples can be seen later in the technical portion of the report. Once this had been done the text, Crafting a Compiler by Fischer and LeBlanc, which is also written in Ada, enabled the writer with the addition of some other materials to produce and carry out the course in the Fall of 1992 as CSC 397 Compiler Design using Ada. The writer, having become somewhat enthusiastic about the use of Ada, is again teaching the Compiler Design course, this semster (spring 1993) using Ada. It might be added that there were start-up difficulties in that the Ada compilers were not at first easily available to the students.

The compiler which we first proposed to use was the Meridian open Ada compiler. This worked well on individual 286, 386 and 486 PC's. However, at Lehigh, the facilities provided to the students are only through networking. That is all of the PC's are networked through LAN's to file servers using Novell software. Unfortunately, license agreements would not permit us to give the students individual copies of the compilers and the software in our LAN's had an unfortunate interaction with the Meridian compilers. The interaction was of such a nature that it made the compilers very difficult to use and the cause of the interaction was not well understood. Fortunately, Lehigh had also made a purchase of IBM Risc/6000 work stations and with those came an AIX Ada compiler. Once again this compiler would not work on the LAN's connecting those systems however it did work on a central machine that had been set up as a mainframe. Though this machine provided something of a bottleneck and the compiler was still inconvenient it was possible for the students to do their work. Later we were

able to make the Meridian compilers more usuable but they were still inconvenient. The computing center now promises to fix the problem on our PC LAN's. However presently the Meridian compliers are now more easily available on our LAN's by using other systems not originally developed by Meridian.

Comments on Course Materials and Syllabus

For the most part the text [F - L] listed below was used as reading material for the students. However some parts of the text, in particular material on the theory of syntax was not quite to the writers liking. Since the formulation of the syntax for the target language had occupied more of the writer's time than anticipated there was less time to produce note materials to fill what were considered to be gaps in description. However, another text, [T - S], was used as a supplement for the required materials. In addition, in the development of the compiler, the writer provided students with partially completed portions and had the students, in teams, complete them. The syllabus of the course was substantially as provided in the project proposal namely:

1) Language constructs and practical implementation of an LL(1) parser.

2) Data structures including arrays and records.

3) Control structures; however, the case statement was omitted.

4) Program structures including procedures and a simplified version of packages.

5) Code generation into the code of an interpretive machine akin to the Pascal p-machine which we call an a-machine.

6) A concrete a-machine written in Ada.

7) An LL(1) syntax for our language.

8) A 4-pass compiler for our language

9) Some discussion of attribute methods in relation to our compiler.

Much of the material was supported by the references below and lectures by the writer. The material from [F - L] was used as follows:

Chap. 2 A brief overview is given. A good part of the general structure of a compiler is in this chapter.

Chap. 3 The scanning touches on the general notion of FSA. The students have had this in another course. Pass1 was discussed with care.

Chap. 5 This chapter deals with syntax and parsing. The material from [T - S] was preferred. See below.

Chap. 7 Semantic processing. This chapter is somewhat cursory. Parts of pass2 were used as

a text. The portions shown to the students will depend on the instructor and the course.

Chap. 8 Symbol tables. This was supplemented by [T - S] see below.

Chap. 9 Run-time storage organization. We preferred [T - S] for this.

Chapters used in [T - S]

Chap. 2, 6 Give a good account of syntax and top down parsing.

Chap. 9 This is a nice account of run time storage organization.

Chap. 11 This was used for semantic analysis and code generation. Parts of pass3 were also used.

Chapters 2, 6 and 9 where used in [T - S]. [H] and [W] provided general inspiration.

[H] is a very practical account of compiling. Unfortunately the book is out of print. However some of the lectures were based on the ideas given there.

References

[F - L] Fischer, Charles N. and R. J. LeBlanc, Jr. Crafting a Compiler, The Benjamin/Cummings Publishing Co., Menlo Park, CA, 1988

[T - S] J-T. Tremblay and P. G. Sorensen, The Theory and Practice of Compiler Writing, McGraw-Hill Book Co. New York, NY, 1985.

[H] P. B. Hansen, Programming a Personal Computer, Prentice-Hall, Inc., Englewood Cliffs, NJ 1982

[W] The Pacals Compiler developed by N. Wirth.

The Compiler:
General Considerations

1. Syntax

Here we present the syntax of the language to be compiled. The syntax is LL(1) and the language has strong similarities to Ada. Some differences were introduced to make sure the user notes the distinction. We use 'proc', 'int', 'bool', 'pack' for 'procedure', 'integer', 'boolean' and 'package' respectively. We do not provide the facility for the procedure name to be used at the end of the procedure and we end our packs with end pack. Here is an example, a bubble sort written in a somewhat complicated way, to demonstrate some of the features of our language. This example will appear again later.

```
proc tst24 is
type table is array(0 .. 25) of int;
type lst is record
            fin: int;
            list: table;
          end record;
i: int;
llst: lst;

pack sort is
  proc init(ll: out lst; n: int);
  proc print(ll: out lst; n: int);
  proc bubsrt(ll: out lst);
end pack;

pack body sort is
   proc init(ll: out lst; n: int) is
   i: int;
```

```
begin
  ll.fin := n;
  i := 1;
  while i <= n loop
   ll.list(i) := n + 1 - i;
   i := i + 1;
  end loop;
end;

proc print(ll: out lst; n: int) is
i: int;
begin
  i := 1;
  while i <= n loop
   write(ll.list(i):4);
   i := i + 1;
  end loop;
end;

proc bubsrt(ll: out lst) is
temp, i, j: int;
begin
  i := 2;
  while i <= ll.fin loop
   ll.list(0) := ll.list(i);
   j := i;
   while ll.list(j) < ll.list(j-1) loop
     temp := ll.list(j);
     ll.list(j) := ll.list(j - 1);
     ll.list(j - 1) := temp;
     j := j - 1;
   end loop;
   i := i + 1;
  end loop;
```

```
  end;
end pack;

begin
  sort.init(llst, 8);
  sort.bubsrt(llst);
  sort.print(llst, 8);
end;
```

The reader will note that this program has an appearance which is substantially like that of Ada. The syntax of our language Mic follows. Note that Λ is the null string and that the syntax has not been 'massaged' into EBNF form. The syntax was created to be very modular. The recursion created by the syntax does not place a very large overhead on execution during parsing. These days with PC's having memories with a minimum of 640k and relatively fast execution the use of syntax in pure BNF form provides no problem. It has the additional advantage that the structure of the parser can be made to reflect the syntax well. This makes program development and debugging relatively simple.

```
terminals
  colon, semicolon, type_id, type, array, is, int, bool, char,
  lparen, rparen, of, int_lit, dotdot, end, record, id,
  comma, eq, ne, lt, le, gt, ge, plus, minus, and, star, slash,
  mod, div, or, not, null, rec_id, proc_id, if, then, elsif, else,
  exit, when, loop,  while , procedure , in, out, gr_char, begin,
  dot, becomes, pkg_id,  pkg_id1,
  pack, body, true, false, write, writeln, read ;

nonterminals
    <declaration>, <object_decl>, <type_decl>, <subprog_decl>,
     <package_decl>, <id_list>,
    <id_list_tail>,  <type_inf>, <type_def>, <stype_def>,
    <array_def>, <index_bd>, <record_def>, <index_range>, <comp_def>,
    <comp_def_tail>, <subprog_header> ,
    <formal_part> , <subprog_part> , <param_decl> , <sparam_decl>, <p_tail> ,
```

<mode> , <p_dec_tail> , <decl_part> , <pkg_tail>, <pkg_def_decl>,
<pkg_d_decl>, <pkg_body_decl>, <pkg_b_decl>,
<relational_op>, <unary_add_op>, <binary_add_op>, <term>, <mul_op>,
<simp_expr>, <sexp_tail>, <factor>, <term_tail>,
<vbl1>, <vbl_tail>, <v1_tail>, <vbl2>, <vbl>, <expression>,
<exp_tail>, <statement>, <lstatement>,
<stat_seq>, <lstat_seq>, <stat_seq_tail>, <lstat_seq_tail>,
<simp_stat>, <write_stat>, <write_body>, <w_tail>,
<read_stat>, <comp_stat>, <proc_call_stat>, <pk_stat>, <pk_stat_tail>,
<assign_stat>, <exit_stat> , <exp_p_sub>,
<p_call_tail>, <p_sub_tail> , <if_stat>, <loop_stat>, <while_stat>,
<elsif_part> , <else_part> , <program>;

start symbol
<program> ;

productions

<declaration> ---> <object_decl>
<declaration> ---> <type_decl>
<declaration> ---> <subprog_decl>
<declaration> ---> <package_decl>
<object_decl> ---> <id_list> colon type_id semicolon
<id_list> ---> id <id_list_tail>
<id_list_tail> ---> comma id <id_list_tail>
<id_list_tail> ---> Λ
<type_ind> ---> type_id
<type_ind> ---> <stype_def>
<type_decl> ---> type type_id is <type_def> semicolon
<type_def> ---> <stype_def>
<stype_def> ---> int
<stype_def> ---> bool
<stype_def> ---> char
<type_def> ---> <array_def>
<type_def> ---> <record_def>

<array_def> ---> array lparen <index_range> rparen of type_id

<index_range> ---> <index_bd> dotdot <index_bd>

<index_bd> ---> int_lit

<index_bd> ---> plus int_lit

<index_bd> ---> minus int_lit

<record_def> ---> record <comp_def> end record

<comp_def> ---> <object_decl> <comp_def_tail>

<comp_def_tail> ---> <object_decl> <comp_def_tail>

<comp_def_tail> ---> Λ

<subprog_decl> ---> <subprog_header> is <subprog_part>
semicolon

<subprog_header> ---> procedure id <formal_part>

<formal_part> ---> lparen <param_decl> rparen

<formal_part> ---> Λ

<sparam_decl> ---> id <p_tail> colon <mode> <type_def>

<p_tail> ---> comma id <p_tail>

<p_tail> ---> Λ

<mode> ---> in

<mode> ---> out

<mode> ---> Λ

<param_decl> ---> <sparam_decl> <p_dec_tail>

<p_dec_tail> ---> semicolon <sparam_decl> <p_dec_tail>

<p_dec_tail> ---> Λ

<subprog_part> ---> <decl_part> begin <stat_seq> end

<decl_part> ---> <declaration> <decl_part>

<decl_part> ---> Λ

<package_decl> ---> pack <pkg_tail>

<pkg_tail> ---> pkg_id is <pkg_def_decl>

<pkg_tail> ---> body pkg_id is <pkg_body_decl>

<pkg_def_decl> ---> <pkg_d_decl> end pack semicolon

<pkg_d_decl> ---> <object_decl> <pkg_d_decl>

<pkg_d_decl> ---> <type_decl> <pkg_d_decl>

<pkg_d_decl> ---> <subprog_header> semicolon <pkg_d_decl>

<pkg_d_decl> ---> Λ

<pkg_body_decl> ---> <pkg_b_decl> end pack semicolon

<pkg_b_decl> ---> <object_decl> <pkg_b_decl>

<pkg_b_decl> ---> <type_decl> <pkg_b_decl>

<pkg_b_decl> ---> <subprog_decl> <pkg_b_decl>

<pkg_b_decl> ---> Λ

<relational_op> ---> eq

<relational_op> ---> ne

<relational_op> ---> lt

<relational_op> ---> le

<relational_op> ---> gt

<relational_op> ---> ge

<unary_add_op> ---> plus

<unary_add_op> ---> minus

<unary_add_op> ---> Λ

<binary_add_op> ---> plus

<binary_add_op> ---> minus

<binary_add_op> ---> or

<mul_op> ---> star

<mul_op> ---> div

<mul_op> ---> mod

<mul_op> ---> and

<term> ---> <factor> <term_tail>

<term_tail> ---> <mul_op> <factor> <term_tail>

<term_tail> ---> Λ

<factor> ---> int_lit

<factor> ---> lparen <expression> rparen

<factor> ---> <vbl>

<factor> ---> not <factor>

<factor> ---> true

<factor> ---> false

<vbl1> ---> rec_id <v1_tail>

<vbl1> ---> id <vbl_tail> <v1_tail>

<v1_tail> ---> dot <vbl1>

<v1_tail> ---> Λ

<vbl_tail> ---> lparen <expression> rparen <vbl_tail>

<vbl_tail> ---> Λ

<vbl2> ---> pkg_id dot <vbl1>

<vbl> ---> <vbl1>

<vbl> ---> <vbl2>

<simp_expr> ---> <unary_add_op> <term> <sexp_tail>

<sexp_tail> ---> <binary_add_op> <term> <sexp_tail>

<sexp_tail> ---> Λ

<expression> ---> <simp_expr> <exp_tail>

<exp_tail> ---> <relational_op> <simp_expr>

<exp_tail> ---> Λ

<lstatement> ---> <statement>

<lstatement> ---> <exit_stat> semicolon

<exit_stat> ---> exit when <expression>

<lstat_seq> ---> <lstatement> <lstat_seq_tail>

<lstat_seq_tail> ---> <lstatement> <lstat_seq_tail>

<lstat_seq_tail> ---> Λ

<statement> ---> <simp_stat> semicolon

<statement> ---> <comp_stat> semicolon

<stat_seq> ---> <statement> <stat_seq_tail>

<stat_seq_tail> ---> <statement> <stat_seq_tail>

<stat_seq_tail> ---> Λ

<simp_stat> ---> null

<simp_stat> ---> <pk_stat>

<pk_stat> ---> pkg_id dot <pk_stat_tail>

<pk_stat_tail> ---> <proc_call_stat>

<pk_stat_tail> ---> <assign_stat>

<simp_stat> ---> <proc_call_stat>

<simp_stat> ---> <assign_stat>

<simp_stat> ---> <write_stat>

<simp_stat> ---> <read_stat>

<write_stat> ---> write <write_body>

<write_stat> ---> writeln

<write_body> ---> lparen <expression> <w_tail> rparen

<w_tail> ---> colon <expression>

<w_tail> ---> Λ

<read_stat> ---> read lparen id rparen

<proc_call_stat> ---> proc_id <p_call_tail>

<exp_p_sub> ---> <expression> <p_sub_tail>

<p_call_tail> ---> lparen <exp_p_sub> rparen

<p_call_tail> ---> Λ

<p_sub_tail> ---> comma <exp_p_sub>

<p_sub_tail> ---> Λ

<assign_stat> ---> <vbl1> becomes <expression>

<comp_stat> ---> <if_stat>

<comp_stat> ---> <loop_stat>

<comp_stat> ---> <while_stat>

<if_stat> ---> if <expression> then <stat_seq> <elsif_part> <else_part>
end if

<elsif_part> ---> elsif <expression> then <stat_seq>

<elsif_part> ---> Λ

<else_part> ---> else <stat_seq>

<else_part> ---> Λ

<loop_stat> ---> loop <lstat_seq> end loop

<while_stat> ---> while <expression> <loop_stat>

<program> ---> <subprog_decl>

In order to be able to use the syntax to write a parser we need an anlaysis of the syntax into first symbols and follow symbols. This is provided by a system written some time ago by the writer. The report of this system is as follows:

NULLABLE NONTERMINALS

<id_list_tail> <comp_def_tail> <formal_part> <p_tail> <mode> <p_dec_tail>
<decl_part> <pkg_d_decl> <pkg_b_decl> <unary_add_op> <sexp_tail>
<term_tail> <vbl_tail> <v1_tail> <exp_tail> <stat_seq_tail> <lstat_seq_tail>
<w_tail> <p_call_tail> <p_sub_tail> <elsif_part> <else_part>

FIRST SETS

<declaration>	type id procedure pack
<object_decl>	id
<type_decl>	type
<subprog_decl>	procedure
<package_decl>	pack
<id_list>	id
<id_list_tail>	Λ comma
<type_ind>	type_id int bool char
<type_def>	array int bool char record
<stype_def>	int bool char
<array_def>	array
<index_bd>	int_lit plus minus
<record_def>	record
<index_range>	int_lit plus minus
<comp_def>	id
<comp_def_tail>	Λ id
<subprog_header>	procedure
<formal_part>	Λ lparen
<subprog_part>	type id procedure begin pack
<param_decl>	id
<sparam_decl>	id
<p_tail>	Λ comma
<mode>	Λ in out
<p_dec_tail>	Λ semicolon
<decl_part>	Λ type id procedure pack
<pkg_tail>	pkg_id body
<pkg_def_decl>	type end id procedure
<pkg_d_decl>	Λ type id procedure
<pkg_body_decl>	type end id procedure
<pkg_b_decl>	Λ type id procedure
<relational_op>	eq ne lt le gt ge

<unary_add_op>	Λ plus minus
<binary_add_op>	plus minus or
<term>	lparen int_lit id not rec_id pkg_id true false
<mul_op>	and star mod div
<simp_expr>	Λ lparen int_lit id plus minus not rec_id pkg_id true false
<sexp_tail>	Λ plus minus or
<factor>	lparen int_lit id not rec_id pkg_id true false
<term_tail>	Λ and star mod div
<vbl1>	id rec_id
<vbl_tail>	Λ lparen
<v1_tail>	Λ dot
<vbl2>	pkg_id
<vbl>	id rec_id pkg_id
<expression>	Λ lparen int_lit id plus minus not rec_id pkg_id true false
<exp_tail>	Λ eq ne lt le gt ge
<statement>	id null rec_id proc_id if loop while pkg_id write writeln read
<lstatement>	id null rec_id proc_id if exit loop while pkg_id write writeln read
<stat_seq>	id null rec_id proc_id if loop while pkg_id write writeln read
<lstat_seq>	id null rec_id proc_id if exit loop while pkg_id write writeln read
<stat_seq_tail>	Λ id null rec_id proc_id if loop while pkg_id write writeln read
<lstat_seq_tail>	Λ id null rec_id proc_id if exit loop while pkg_id write writeln read
<simp_stat>	id null rec_id proc_id pkg_id write writeln read
<write_stat>	write writeln
<write_body>	lparen
<w_tail>	Λ colon

<read_stat>	read
<comp_stat>	if loop while
<proc_call_stat>	proc_id
<pk_stat>	pkg_id
<pk_stat_tail>	id rec_id proc_id
<assign_stat>	id rec_id
<exit_stat>	exit
<exp_p_sub>	Λ lparen int_lit id plus minus not rec_id pkg_id true false
<p_call_tail>	Λ lparen
<p_sub_tail>	Λ comma
<if_stat>	if
<loop_stat>	loop
<while_stat>	while
<elsif_part>	Λ elsif
<else_part>	Λ else
<program>	procedure

FOLLOW SETS

<declaration>	type id procedure begin pack
<object_decl>	type end id procedure begin pack
<type_decl>	type end id procedure begin pack
<subprog_decl>	Λ type end id procedure begin pack
<package_decl>	type id procedure begin pack
<id_list>	colon
<id_list_tail>	colon
<type_ind>	semicolon rparen
<type_def>	semicolon
<stype_def>	semicolon rparen
<array_def>	semicolon
<index_bd>	rparen dotdot
<record_def>	semicolon
<index_range>	rparen

<comp_def>	end
<comp_def_tail>	end
<subprog_header>	semicolon is
<formal_part>	semicolon is
<subprog_part>	semicolon
<param_decl>	rparen
<sparam_decl>	semicolon rparen
<p_tail>	colon
<mode>	array int bool char record
<p_dec_tail>	rparen
<decl_part>	begin
<pkg_tail>	type id procedure begin pack
<pkg_def_decl>	type id procedure begin pack
<pkg_d_decl>	end
<pkg_body_decl>	type id procedure begin pack
<pkg_b_decl>	end
<relational_op>	semicolon lparen rparen int_lit id comma plus minus not rec_id then loop pkg_id true false
<unary_add_op>	lparen int_lit id not rec_id pkg_id true false
<binary_add_op>	lparen int_lit id not rec_id pkg_id true false
<term>	colon semicolon rparen comma eq ne lt le gt ge plus minus or then loop
<mul_op>	lparen int_lit id not rec_id pkg_id true false
<simp_expr>	colon semicolon rparen comma eq ne lt le gt ge then loop
<sexp_tail>	colon semicolon rparen comma eq ne lt le gt ge then loop
<factor>	colon semicolon rparen comma eq ne lt le gt ge plus minus and star mod div or then loop
<term_tail>	colon semicolon rparen comma eq ne lt le gt ge plus minus or then loop
<vbl1>	colon semicolon rparen comma eq ne lt le gt ge plus minus and star mod div or then loop

	becomes
<vbl_tail>	colon semicolon rparen comma eq ne lt le gt ge plus minus and star mod div or then loop dot becomes
<v1_tail>	colon semicolon rparen comma eq ne lt le gt ge plus minus and star mod div or then loop becomes
<vbl2>	colon semicolon rparen comma eq ne lt le gt ge plus minus and star mod div or then loop
<vbl>	colon semicolon rparen comma eq ne lt le gt ge plus minus and star mod div or then loop
<expression>	colon semicolon rparen comma then loop
<exp_tail>	colon semicolon rparen comma then loop
<statement>	end id null rec_id proc_id if elsif else exit loop while pkg_id write writeln read
<lstatement>	end id null rec_id proc_id if exit loop while pkg_id write writeln read
<stat_seq>	end elsif else
<lstat_seq>	end
<stat_seq_tail>	end elsif else
<lstat_seq_tail>	end
<simp_stat>	semicolon
<write_stat>	semicolon
<write_body>	semicolon
<w_tail>	rparen
<read_stat>	semicolon
<comp_stat>	semicolon
<proc_call_stat>	semicolon
<pk_stat>	semicolon
<pk_stat_tail>	semicolon
<assign_stat>	semicolon
<exit_stat>	semicolon
<exp_p_sub>	rparen
<p_call_tail>	semicolon

<p_sub_tail>	rparen
<if_stat>	semicolon
<loop_stat>	semicolon
<while_stat>	semicolon
<elsif_part>	end else
<else_part>	end
<program>	Λ

CHECKING FOR LL(1) STRUCTURE

SYNTAX IS LL(1)

FIRSTSETS OF RIGHT SIDES OF PRODUCTIONS

1: <declaration> ---> <object_decl>
First set is [id]
2: <declaration> ---> <type_decl>
First set is [type]
3: <declaration> ---> <subprog_decl>
First set is [procedure]
4: <declaration> ---> <package_decl>
First set is [pack]
5: <object_decl> ---> <id_list> colon <type_ind> semicolon
First set is [id]
6: <type_decl> ---> type type_id is <type_def> semicolon
First set is [type]
7: <subprog_decl> ---> <subprog_header> is <subprog_part> semicolon
First set is [procedure]
8: <package_decl> ---> pack <pkg_tail>
First set is [pack]
9: <id_list> ---> id <id_list_tail>
First set is [id]
10: <id_list_tail> ---> comma id <id_list_tail>

First set is [comma]

11: <id_list_tail> ---> /\

First set is [lambda1]

12: <type_ind> ---> type_id

First set is [type_id]

13: <type_ind> ---> <stype_def>

First set is [int,bool,char]

14: <type_def> ---> <stype_def>

First set is [int,bool,char]

15: <type_def> ---> <array_def>

First set is [array]

16: <type_def> ---> <record_def>

First set is [record]

17: <stype_def> ---> int

First set is [int]

18: <stype_def> ---> bool

First set is [bool]

19: <stype_def> ---> char

First set is [char]

20: <array_def> ---> array lparen <index_range> rparen of <type_ind>

First set is [array]

21: <index_bd> ---> int_lit

First set is [int_lit]

22: <index_bd> ---> plus int_lit

First set is [plus]

23: <index_bd> ---> minus int_lit

First set is [minus]

24: <record_def> ---> record <comp_def> end record

First set is [record]

25: <index_range> ---> <index_bd> dotdot <index_bd>

First set is [int_lit,plus,minus]

26: <comp_def> ---> <object_decl> <comp_def_tail>

First set is [id]

27: <comp_def_tail> ---> <object_decl> <comp_def_tail>

First set is [id]

28: <comp_def_tail> ---> /\

First set is [lambda1]

29: <subprog_header> ---> procedure id <formal_part>

First set is [procedure]

30: <formal_part> ---> lparen <param_decl> rparen

First set is [lparen]

31: <formal_part> ---> /\

First set is [lambda1]

32: <subprog_part> ---> <decl_part> begin <stat_seq> end

First set is [type,id,procedure,begin,pack]

33: <param_decl> ---> <sparam_decl> <p_dec_tail>

First set is [id]

34: <sparam_decl> ---> id <p_tail> colon <mode> <type_ind>

First set is [id]

35: <p_tail> ---> comma id <p_tail>

First set is [comma]

36: <p_tail> ---> /\

First set is [lambda1]

37: <mode> ---> in

First set is [in]

38: <mode> ---> out

First set is [out]

39: <mode> ---> /\

First set is [lambda1]

40: <p_dec_tail> ---> semicolon <sparam_decl> <p_dec_tail>

First set is [semicolon]

41: <p_dec_tail> ---> /\

First set is [lambda1]

42: <decl_part> ---> <declaration> <decl_part>

First set is [type,id,procedure,pack]

43: <decl_part> ---> /\

First set is [lambda1]

44: <pkg_tail> ---> pkg_id is <pkg_def_decl>

First set is [pkg_id]

45: <pkg_tail> ---> body pkg_id is <pkg_body_decl>

First set is [body]

46: <pkg_def_decl> ---> <pkg_d_decl> end pack semicolon

First set is [type,end,id,procedure]

47: <pkg_d_decl> ---> <object_decl> <pkg_d_decl>

First set is [id]

48: <pkg_d_decl> ---> <type_decl> <pkg_d_decl>

First set is [type]

49: <pkg_d_decl> ---> <subprog_header> semicolon <pkg_d_decl>

First set is [procedure]

50: <pkg_d_decl> ---> /\

First set is [lambda1]

51: <pkg_body_decl> ---> <pkg_b_decl> end pack semicolon

First set is [type,end,id,procedure]

52: <pkg_b_decl> ---> <object_decl> <pkg_b_decl>

First set is [id]

53: <pkg_b_decl> ---> <type_decl> <pkg_b_decl>

First set is [type]

54: <pkg_b_decl> ---> <subprog_decl> <pkg_b_decl>

First set is [procedure]

55: <pkg_b_decl> ---> /\

First set is [lambda1]

56: <relational_op> ---> eq

First set is [eq]

57: <relational_op> ---> ne

First set is [ne]

58: <relational_op> ---> lt

First set is [lt]

59: <relational_op> ---> le

First set is [le]

60: <relational_op> ---> gt

First set is [gt]

61: <relational_op> ---> ge

First set is [ge]

62: <unary_add_op> ---> plus

First set is [plus]

63: <unary_add_op> ---> minus

First set is [minus]

64: <unary_add_op> ---> /\

First set is [lambda1]

65: <binary_add_op> ---> plus

First set is [plus]

66: <binary_add_op> ---> minus

First set is [minus]

67: <binary_add_op> ---> or

First set is [or]

68: <term> ---> <factor> <term_tail>

First set is [lparen,int_lit,id,not,rec_id,pkg_id,true,false]

69: <mul_op> ---> star

First set is [star]

70: <mul_op> ---> div

First set is [div]

71: <mul_op> ---> mod

First set is [mod]

72: <mul_op> ---> and

First set is [and]

73: <simp_expr> ---> <unary_add_op> <term> <sexp_tail>

First set is [lparen,int_lit,id,plus,minus,not,rec_id,pkg_id,true,
false]

74: <sexp_tail> ---> <binary_add_op> <term> <sexp_tail>

First set is [plus,minus,or]

75: <sexp_tail> ---> /\

First set is [lambda1]

76: <factor> ---> int_lit

First set is [int_lit]

77: <factor> ---> lparen <expression> rparen

First set is [lparen]

78: <factor> ---> <vbl>

First set is [id,rec_id,pkg_id]

79: <factor> ---> not <factor>

First set is [not]

80: <factor> ---> true

First set is [true]

81: <factor> ---> false

First set is [false]

82: <term_tail> ---> <mul_op> <factor> <term_tail>

First set is [and,star,mod,div]

83: <term_tail> ---> /\

First set is [lambda1]

84: <vbl1> ---> rec_id <v1_tail>

First set is [rec_id]

85: <vbl1> ---> id <vbl_tail> <v1_tail>

First set is [id]

86: <vbl_tail> ---> lparen <expression> rparen <vbl_tail>

First set is [lparen]

87: <vbl_tail> ---> /\

First set is [lambda1]

88: <v1_tail> ---> dot <vbl1>

First set is [dot]

89: <v1_tail> ---> /\

First set is [lambda1]

90: <vbl2> ---> pkg_id dot <vbl1>

First set is [pkg_id]

91: <vbl> ---> <vbl1>

First set is [id,rec_id]

92: <vbl> ---> <vbl2>

First set is [pkg_id]

93: <expression> ---> <simp_expr> <exp_tail>

First set is [lambda1,lparen,int_lit,id,plus,minus,not,rec_id,pkg_id,
true,false]

94: <exp_tail> ---> <relational_op> <simp_expr>

First set is [eq,ne,lt,le,gt,ge]

95: <exp_tail> ---> /\

First set is [lambda1]

96: <statement> ---> <simp_stat> semicolon

First set is [id,null,rec_id,proc_id,pkg_id,write,writeln,read]

97: <statement> ---> <comp_stat> semicolon

First set is [if,loop,while]

98: <lstatement> ---> <statement>

First set is [id,null,rec_id,proc_id,if,loop,while,pkg_id,write,
writeln,read]

99: <lstatement> ---> <exit_stat> semicolon

First set is [exit]

100: <stat_seq> ---> <statement> <stat_seq_tail>

First set is [id,null,rec_id,proc_id,if,loop,while,pkg_id,write,
writeln,read]

101: <lstat_seq> ---> <lstatement> <lstat_seq_tail>

First set is [id,null,rec_id,proc_id,if,exit,loop,while,pkg_id,
write,writeln,read]

102: <stat_seq_tail> ---> <statement> <stat_seq_tail>

First set is [id,null,rec_id,proc_id,if,loop,while,pkg_id,write,
writeln,read]

103: <stat_seq_tail> ---> /\

First set is [lambda1]

104: <lstat_seq_tail> ---> <lstatement> <lstat_seq_tail>

First set is [id,null,rec_id,proc_id,if,exit,loop,while,pkg_id,
write,writeln,read]

105: <lstat_seq_tail> ---> /\

First set is [lambda1]

106: <simp_stat> ---> null

First set is [null]

107: <simp_stat> ---> <pk_stat>

First set is [pkg_id]

108: <simp_stat> ---> <proc_call_stat>

First set is [proc_id]

109: <simp_stat> ---> <assign_stat>

First set is [id,rec_id]

110: <simp_stat> ---> <write_stat>

First set is [write,writeln]

111: <simp_stat> ---> <read_stat>

First set is [read]

112: <write_stat> ---> write <write_body>

First set is [write]

113: <write_stat> ---> writeln

First set is [writeln]

114: <write_body> ---> lparen <expression> <w_tail> rparen

First set is [lparen]

115: <w_tail> ---> colon <expression>

First set is [colon]

116: <w_tail> ---> /\

First set is [lambda1]

117: <read_stat> ---> read lparen id rparen

First set is [read]

118: <comp_stat> ---> <if_stat>

First set is [if]

119: <comp_stat> ---> <loop_stat>

First set is [loop]

120: <comp_stat> ---> <while_stat>

First set is [while]

121: <proc_call_stat> ---> proc_id <p_call_tail>

First set is [proc_id]

122: <pk_stat> ---> pkg_id dot <pk_stat_tail>

First set is [pkg_id]

123: <pk_stat_tail> ---> <proc_call_stat>

First set is [proc_id]

124: <pk_stat_tail> ---> <assign_stat>

First set is [id,rec_id]

125: <assign_stat> ---> <vbl1> becomes <expression>

First set is [id,rec_id]

126: <exit_stat> ---> exit when <expression>

First set is [exit]

127: <exp_p_sub> ---> <expression> <p_sub_tail>

First set is [lambda1,lparen,int_lit,id,plus,minus,not,rec_id,pkg_id,
true,false]

128: <p_call_tail> ---> lparen <exp_p_sub> rparen

First set is [lparen]

129: <p_call_tail> ---> /\

First set is [lambda1]

130: <p_sub_tail> ---> comma <exp_p_sub>

First set is [comma]

131: <p_sub_tail> ---> /\

First set is [lambda1]

132: <if_stat> ---> if <expression> then <stat_seq> <elsif_part> <else_part> end if

First set is [if]

133: <loop_stat> ---> loop <lstat_seq> end loop

First set is [loop]

134: <while_stat> ---> while <expression> <loop_stat>

First set is [while]

135: <elsif_part> ---> elsif <expression> then <stat_seq>

First set is [elsif]

136: <elsif_part> ---> /\

First set is [lambda1]

137: <else_part> ---> else <stat_seq>

First set is [else]

138: <else_part> ---> /\

First set is [lambda1]

139: <program> ---> <subprog_decl>

First set is [procedure]

2. Pass 1: Lexical Analysis.

This pass reflects several important choices made for the ultimate operation of the compiler. First and foremost the compiler was developed for pedagogical purposes and was not intended to become a production compiler (although the latter goal could be offered as a future student project.) This means that the error reporting system would be quite primitive. Second, since compilers are quite complex simplicity was made a major watchword. Thus for example we chose linear search rather than hashing. Further we decided not to maintain a printname table for error reporting. Error reporting could be done simply by indicating the line the error is on. In fact an extensive error diagnosis sytem was implemented but this was intended for use by the implementer as a debugging tool. However, this diagnostic tool could eventually be turned into a method for reporting errors to the user. Thus, all identifiers are turned into internal symbols and the later passes do not know the print name. Pass1 produces two files. These are out1 which is used by Pass2 and mnem1 which is intended to be used by the implementer. mnem1 is a mnemonic version of out1. The production of mnem1 can of course be disabled. Pass1 requests the name of the source file and then produces the above two files. Since pass1 cannot do any syntax analysis there are very few errors which it can report. About the only error of consequence to the lexical analysis is an illegal symbol in the syntax. The files out1 and mnem1 for the bubble sort program presented above look like this:

out1:

```
19 1 51 7 57 39 19 2 40 7 58 39 35 9 8 0 38 8 25 10 44 33 11 19 3 40 7 59
39 48 19 4 7 60 32 33 11 19 5 7 61 32 7 58 11 19 6 1 48 11 19 7 7 62 32
33 11 19 8 7 63 32 7 59 11 19 9 19 10 47 7 64 39 19 11 51 7 65 9 7 66 32
14 7 59 11 7 67 32 33 10 11 19 12 51 7 68 9 7 66 32 14 7 59 11 7 67 32 33
10 11 19 13 51 7 69 9 7 66 32 14 7 59 10 11 19 14 1 47 11 19 15 19 16 47
36 7 64 39 19 17 51 7 65 9 7 66 32 14 7 59 11 7 67 32 33 10 39 19 18 7 62
32 33 11 19 19 0 19 20 7 66 2 7 60 4 7 67 11 19 21 7 62 4 8 1 11 19 22 50
7 62 22 7 67 17 19 23 7 66 2 7 61 9 7 62 10 4 7 67 5 8 1 6 7 62 11 19 24
7 62 4 7 62 5 8 1 11 19 25 1 17 11 19 26 1 11 19 27 19 28 51 7 68 9 7 66
32 14 7 59 11 7 67 32 33 10 39 19 29 7 62 32 33 11 19 30 0 19 31 7 62 4
```

8 1 11 19 32 50 7 62 22 7 67 17 19 33 53 9 7 66 2 7 61 9 7 62 10 32 8 4
10 11 19 34 7 62 4 7 62 5 8 1 11 19 35 1 17 11 19 36 1 11 19 37 19 38 51
7 69 9 7 66 32 14 7 59 10 39 19 39 7 70 12 7 62 12 7 71 32 33 11 19 40 0
19 41 7 62 4 8 2 11 19 42 50 7 62 22 7 66 2 7 60 17 19 43 7 66 2 7 61 9
8 0 10 4 7 66 2 7 61 9 7 62 10 11 19 44 7 71 4 7 62 11 19 45 50 7 66 2 7
61 9 7 71 10 21 7 66 2 7 61 9 7 71 6 8 1 10 17 19 46 7 70 4 7 66 2 7 61
9 7 71 10 11 19 47 7 66 2 7 61 9 7 71 10 4 7 66 2 7 61 9 7 71 6 8 1 10 11
19 48 7 66 2 7 61 9 7 71 6 8 1 10 4 7 70 11 19 49 7 71 4 7 71 6 8 1 11 19
50 1 17 11 19 51 7 62 4 7 62 5 8 1 11 19 52 1 17 11 19 53 1 11 19 54 1 47
11 19 55 19 56 0 19 57 7 64 2 7 65 9 7 63 12 8 8 10 11 19 58 7 64 2 7 69
9 7 63 10 11 19 59 7 64 2 7 68 9 7 63 12 8 8 10 11 19 60 1 11 13

mnem1:

linenol 1 procl idl 57 isl linenol 2 typel idl 58 isl arrayl lparenthl
intlitl 0 dotdotl intlitl 25 rparenthl ofl intl semicolonl linenol
3 typel idl 59 isl recordl linenol 4 idl 60 colonl intl semicolonl
linenol 5 idl 61 colonl idl 58 semicolonl linenol 6 endl recordl
semicolonl linenol 7 idl 62 colonl intl semicolonl linenol 8 idl
63 colonl idl 59 semicolonl linenol 9 linenol 10 packl idl 64 isl
linenol 11 procl idl 65 lparenthl idl 66 colonl outl idl 59 semicolonl
idl 67 colonl intl rparenthl semicolonl linenol 12 procl idl 68 lparenthl
idl 66 colonl outl idl 59 semicolonl idl 67 colonl intl rparenthl
semicolonl linenol 13 procl idl 69 lparenthl idl 66 colonl outl idl
59 rparenthl semicolonl linenol 14 endl packl semicolonl linenol
15 linenol 16 packl bodyl idl 64 isl linenol 17 procl idl 65 lparenthl
idl 66 colonl outl idl 59 semicolonl idl 67 colonl intl rparenthl
isl linenol 18 idl 62 colonl intl semicolonl linenol 19 beginl linenol
20 idl 66 dotl idl 60 becomesl idl 67 semicolonl linenol 21 idl 62
becomesl intlitl 1 semicolonl linenol 22 whilel idl 62 le2 idl 67
loopl linenol 23 idl 66 dotl idl 61 lparenthl idl 62 rparenthl becomesl
idl 67 plusopl intlitl 1 minusopl idl 62 semicolonl linenol 24 idl
62 becomesl idl 62 plusopl intlitl 1 semicolonl linenol 25 endl loopl
semicolonl linenol 26 endl semicolonl linenol 27 linenol 28 procl

id1 68 lparenth1 id1 66 colon1 out1 id1 59 semicolon1 id1 67 colon1
int1 rparenth1 is1 lineno1 29 id1 62 colon1 int1 semicolon1 lineno1
30 begin1 lineno1 31 id1 62 becomes1 intlit1 1 semicolon1 lineno1
32 while1 id1 62 le2 id1 67 loop1 lineno1 33 write1 lparenth1 id1
66 dot1 id1 61 lparenth1 id1 62 rparenth1 colon1 intlit1 4 rparenth1
semicolon1 lineno1 34 id1 62 becomes1 id1 62 plusop1 intlit1 1 semicolon1
lineno1 35 end1 loop1 semicolon1 lineno1 36 end1 semicolon1 lineno1
37 lineno1 38 proc1 id1 69 lparenth1 id1 66 colon1 out1 id1 59 rparenth1
is1 lineno1 39 id1 70 comma1 id1 62 comma1 id1 71 colon1 int1 semicolon1
lineno1 40 begin1 lineno1 41 id1 62 becomes1 intlit1 2 semicolon1
lineno1 42 while1 id1 62 le2 id1 66 dot1 id1 60 loop1 lineno1 43
id1 66 dot1 id1 61 lparenth1 intlit1 0 rparenth1 becomes1 id1 66
dot1 id1 61 lparenth1 id1 62 rparenth1 semicolon1 lineno1 44 id1
71 becomes1 id1 62 semicolon1 lineno1 45 while1 id1 66 dot1 id1 61
lparenth1 id1 71 rparenth1 lt1 id1 66 dot1 id1 61 lparenth1 id1 71
minusop1 intlit1 1 rparenth1 loop1 lineno1 46 id1 70 becomes1 id1
66 dot1 id1 61 lparenth1 id1 71 rparenth1 semicolon1 lineno1 47 id1
66 dot1 id1 61 lparenth1 id1 71 rparenth1 becomes1 id1 66 dot1 id1
61 lparenth1 id1 71 minusop1 intlit1 1 rparenth1 semicolon1 lineno1
48 id1 66 dot1 id1 61 lparenth1 id1 71 minusop1 intlit1 1 rparenth1
becomes1 id1 70 semicolon1 lineno1 49 id1 71 becomes1 id1 71 minusop1
intlit1 1 semicolon1 lineno1 50 end1 loop1 semicolon1 lineno1 51
id1 62 becomes1 id1 62 plusop1 intlit1 1 semicolon1 lineno1 52 end1
loop1 semicolon1 lineno1 53 end1 semicolon1 lineno1 54 end1 pack1
semicolon1 lineno1 55 lineno1 56 begin1 lineno1 57 id1 64 dot1 id1
65 lparenth1 id1 63 comma1 intlit1 8 rparenth1 semicolon1 lineno1
58 id1 64 dot1 id1 69 lparenth1 id1 63 rparenth1 semicolon1 lineno1
59 id1 64 dot1 id1 68 lparenth1 id1 63 comma1 intlit1 8 rparenth1
semicolon1 lineno1 60 end1 semicolon1 endtext1

The code for pass1 is now listed. At this point and for the future we make no apologies for the infelicities of code. Though we have made every effort to maintain good design engineering practices we are certain that some awkward code occurs. However given the fact that we had little time to carefully reexamine the compiler most of the code appears to be

in reasonable shape. As always, understanding code which you have not participated in writing is difficult. We hope the reader will make the necessary effort to see the point our programs.

```
with text_io;
package int_io is new text_io.integer_io(integer);
with text_io; with int_io;
procedure pass1 is
  rsrvtop: constant   := 32;
  symstart: constant := 56;
  endtable: constant  := 55;
  type symbols is (begin1, end1, dot1, elsif1, becomes1, plusop1, --5
                   minusop1,id1, intlit1, lparenth1, rparenth1, --10
                   semicolon1, comma1, endtext1, out1, if1, --15
                   then1, loop1, exit1, lineno1, error1, lt1, le1, --22
                   gt1, ge1, eq1, ne1 , star1, slash1, and1,  --29
                   or1, not1, colon1, int1, bool1,  array1, --35
                   body1, char1, dotdot1, is1, type1, true1, false1,--42
                   else1, of1, mod1, null1, pack1, record1, --48
   when1, while1, proc1, in1, write1, writeln1, read1); --55

  type errors is (numerical2, unknown2);
  nm_error: exception;
  subtype str11 is string(1 .. 11);
  subtype str80 is string(1 .. 80);
  subtype str20 is string(1 .. 20);

  mnem: array(integer range 0..endtable) of str11;
  type symrec is record
                  wd: str11;
                  lnth: integer;
                  val: integer;
                end record;
  symboltable: array(integer range 0 .. 60) of symrec;
  progfile, fout1, listfile, mnfile: text_io.file_type;
```

```
 ch: character;  x: integer;
 line: array(integer range 1 .. 80) of character; cc, lc: integer;
 sym: symbols; idno, lineno: integer;
 outlen, mnemlen, linenum: integer;
 type wordrec is record
                  len: integer;
                  wrd: str80;
                end record;
 word: wordrec; symtop: integer;
 endfile: boolean;

procedure init is
term: symbols; i: integer;
begin
   mnem(0) := "begin1     ";
   mnem(1) := "end1       ";
   mnem(2) := "dot1       ";
   mnem(3) := "elsif1     ";
   mnem(4) := "becomes1   ";
   mnem(5) := "plusop1    ";
   mnem(6) := "minusop1   ";
   mnem(7) := "id1        ";
   mnem(8) := "intlit1    ";
   mnem(9) := "lparenth1  ";
   mnem(10) := "rparenth1  ";
   mnem(11) := "semicolon1 ";
   mnem(12) := "comma1     ";
   mnem(13) := "endtext1   ";
   mnem(14) := "out1       ";
   mnem(15) := "if1        ";
   mnem(16) := "then1      ";
   mnem(17) := "loop1      ";
   mnem(18) := "exit1      ";
   mnem(19) := "lineno1    ";
```

```
mnem(20) := "error1    ";
mnem(21) := "lt1       ";
mnem(22) := "le2       ";
mnem(23) := "gt1       ";
mnem(24) := "ge1       ";
mnem(25) := "eq1       ";
mnem(26) := "ne1       ";
mnem(27) := "star1     ";
mnem(28) := "slash1    ";
mnem(29) := "and1      ";
mnem(30) := "or1       ";
mnem(31) := "not1      ";
mnem(32) := "colon1    ";
mnem(33) := "int1      ";
mnem(34) := "bool1     ";
mnem(35) := "array1    ";
mnem(36) := "body1     ";
mnem(37) := "char1     ";
mnem(38) := "dotdot1   ";
mnem(39) := "is1       ";
mnem(40) := "type1     ";
mnem(41) := "true1     ";
mnem(42) := "false1    ";
mnem(43) := "else1     ";
mnem(44) := "of1       ";
mnem(45) := "mod1      ";
mnem(46) := "null1     ";
mnem(47) := "pack1     ";
mnem(48) := "record1   ";
mnem(49) := "when1     ";
mnem(50) := "while1    ";
mnem(51) := "proc1     ";
mnem(52) := "in1       ";
mnem(53) := "write1    ";
```

```
mnem(54) := "writeln1   ";
mnem(55) := "read1      ";

symboltable(0) :=  ("begin      ",5,  0);
symboltable(1) :=  ("end        ",3,  1);
symboltable(2) :=  ("else       ",4,  43);
symboltable(3) :=  ("elsif      ",5,  3);
symboltable(4) :=  ("out        ",3,  14);
symboltable(5) :=  ("if         ",2,  15);
symboltable(6) :=  ("then       ",4,  16);
symboltable(7) :=  ("loop       ",4,  17);
symboltable(8) :=  ("exit       ",4,  18);
symboltable(9) :=  ("and        ",3,  29);
symboltable(10) := ("or         ",2,  30);
symboltable(11) := ("not        ",3,  31);
symboltable(12) := ("int        ",3,  33);
symboltable(13) := ("bool       ",4,  34);
symboltable(14) := ("array      ",5,  35);
symboltable(15) := ("is         ",2,  39);
symboltable(16) := ("type       ",4,  40);
symboltable(17) := ("true       ",4,  41);
symboltable(18) := ("false      ",5,  42);
symboltable(19) := ("body       ",4,  36);
symboltable(20) := ("char       ",4,  37);
symboltable(21) := ("of         ",2,  44);
symboltable(22) := ("mod        ",3,  45);
symboltable(23) := ("null       ",4,  46);
symboltable(24) := ("pack       ",4,  47);
symboltable(25) := ("record     ",6,  48);
symboltable(26) := ("when       ",4,  49);
symboltable(27) := ("while      ",5,  50);
symboltable(28) := ("proc       ",4,  51);
symboltable(29) := ("in         ",2,  52);
symboltable(30) := ("write      ",5,  53);
```

```
    symboltable(31) := ("writeln    ",7,  54);
    symboltable(32) := ("read       ",4,  55);

    idno := symstart;
    outlen := 0; mnemlen := 0;
    linenum := 0;
    symtop := rsrvtop; endfile := false;
end;


function size(n: integer) return integer is
neg: boolean; nn, len: integer;
begin
  nn := n;
  neg := (nn < 0);
  if neg then nn := -nn; end if;
  if  nn <= 9 then  len := 1;
  elsif nn <= 99 then len := 2;
  elsif nn <= 999 then len := 3;
  elsif nn <= 9999 then len := 4;
  elsif nn <= 32767 then len := 5;
  end if;
  if neg then return len + 1;
  else return len; end if;
end size;

procedure emit(n: integer) is
sz: integer;
begin
  sz := size(n);
  text_io.put(fout1,' '); int_io.put(fout1,n, sz);
  outlen := outlen + sz + 1;
  if outlen >= 72 then
    text_io.put_line(fout1,"");
```

```
    outlen := 0;
  end if ;
end emit;

function length(st: str11) return integer is
i: integer;
begin
  i := 11;
  while st(i) = ' ' loop
    i := i - 1;
  end loop;
  return i;
end;

procedure emit1(k,n: integer) is
lngth, sz: integer;
begin
  if k = 1 then
    lngth := length(mnem(n));
    emit(n);
    text_io.put(mnfile, ' '& mnem(n)(1 .. lngth));
    mnemlen := mnemlen + lngth + 1;
    if mnemlen >= 65 then
      text_io.put_line(mnfile,"");
      mnemlen := 0;
    end if;
  else
     emit(n);
     sz := size(n);
     text_io.put(mnfile,' '); int_io.put(mnfile,n,sz);
     mnemlen := mnemlen + sz + 1;
     if mnemlen >= 65 then
       text_io.put_line(mnfile,"");
       mnemlen := 0;
```

```
    end if;
  end if;
end emit1;


procedure emit2(m,n:integer) is
begin
  emit1(1,m); emit1(2,n);
end;

procedure error(k: integer) is
begin
    emit2(symbols'pos(error1), k);
end;

procedure nextch(kh: out character) is
i : integer;
begin
 if (cc = lc) and text_io.end_of_file(progfile) then
   endfile := true; kh := ' ';
 else
   -- else begins here
   if (cc = lc) and (not text_io.end_of_file(progfile)) then
      i := 0;
      while not text_io.end_of_line(progfile) loop
        i := i + 1;
        text_io.get(progfile, ch);
        text_io.put(listfile, ch);
        if ('A' <= ch) and (ch <= 'Z') then
          line(i) :=
           character'val(character'pos(ch) - character'pos('A')
                      + character'pos('a'));
        else
          line(i) := ch;
```

```
        end if;
      end loop;
      lc := i + 1;
      line(lc) := ' ';
      cc := 0;
      text_io.skip_line(progfile);
      text_io.new_line(listfile);
      linenum := linenum + 1;
      emit2(symbols'pos(lineno1),linenum);
   end if;
  cc := cc + 1;
  kh := line(cc);
  -- else ends here
 end if;
end;


procedure getfile is
use text_io;
subtype str30 is string(1 .. 30);
fn:str30; i: natural;
begin
  put("program file > ");
  get_line(standard_input,fn, i);
  open(progfile,in_file, fn(1 .. i));
  create(listfile,out_file,"lister");
  create(fout1,out_file,"out1");
  create(mnfile,out_file, "mnem1");
  exception
    when name_error =>
     put_line("File " & fn(1 .. i) & " Not Found");
     raise nm_error;
end:
```

```
procedure getnumber(n: integer) is
value: integer;
begin
  value := 0;
  loop
    if (value <= 3276) or ((value = 3276)  and (ch <= '7')) then
      value := 10 * value + character'pos(ch)
            - character'pos('0');
      nextch(ch);
    else
      error(errors'pos(numerical2));
      while ('0' <= ch) and (ch <= '9') loop nextch(ch); end loop;
      value := 0;
    end if;
    exit when (ch < '0') or ('9' < ch);
  end loop;
  emit2(symbols'pos(intlit1), n * value);
end;

procedure getword is
i: integer;
begin
  i := 0;
  loop
    i := i + 1;
    word.wrd(i) := ch;
    nextch(ch);
    exit when not ( (('a' <= ch) and (ch <= 'z')) or
       (('0' <= ch) and (ch <= '9')) );
  end loop;
  if i >= 20 then
    word.len := 20;
  else
    word.len :=i;
```

```
  end if;
end;

function check(oldwd, nuwd: wordrec) return boolean is
i, len: integer; ok: boolean;
begin
  len := nuwd.len;
  if len = oldwd.len then
    i := 1;
    ok := true;
    while ok loop
     ok := oldwd.wrd(i) = nuwd.wrd(i);
     if ok then
       i := i + 1;
       ok := i <= len;
     end if;
   end loop;
   return i > len;
  else return false;
  end if;
end;

procedure insert(j: integer) is
i: integer;
begin
  for i in 1 .. word.len loop
    symboltable(j).wd(i) := word.wrd(i);
  end loop;
  symboltable(j).lnth := word.len;
  idno := idno + 1;
  symboltable(j).val := idno;
end;

procedure findword is
```

```
i,j, k: integer; oldwd: wordrec;
begin
 j := symtop;
 j := j + 1;
 insert(j);
 i := -1;
 loop
  i := i + 1;
  oldwd.len := symboltable(i).lnth;
  for k in 1 .. oldwd.len loop
   oldwd.wrd(k) := symboltable(i).wd(k);
  end loop;
 exit when check(oldwd, word);
 end loop;
 if i <= rsrvtop then
  emit1(1,symboltable(i).val);
 else
  emit2(symbols'pos(id1), symboltable(i).val);
 end if;
 if i < j then
  idno := idno - 1;
 else
  symtop := j;
 end if;
end;

procedure nextsym is
begin
 while (ch = ' ') and (not endfile) loop nextch(ch); end loop;
 if not endfile then
  if ('a' <= ch) and (ch <= 'z') then
   getword;
   findword;
  elsif ('0' <= ch) and (ch <= '9') then
```

```
      getnumber(1);
    else
      case ch is
        when ',' => emit1(1,symbols'pos(comma1)); nextch(ch);
        when ';' => emit1(1,symbols'pos(semicolon1)); nextch(ch);
        when '(' => emit1(1,symbols'pos(lparenth1)); nextch(ch);
        when ')' => emit1(1,symbols'pos(rparenth1)); nextch(ch);
        when '+' => emit1(1,symbols'pos(plusop1)); nextch(ch);
        when '*' => emit1(1,symbols'pos(star1)); nextch(ch);
when '-' => emit1(1,symbols'pos(minusop1)); nextch(ch);
when '/' =>
              nextch(ch);
              if ch = '=' then
                emit1(1,symbols'pos(ne1));
                nextch(ch);
              else
                emit1(1,symbols'pos(slash1));
              end if;
        when ':' =>
              nextch(ch);
              if ch = '=' then
                emit1(1,symbols'pos(becomes1));
                nextch(ch);
              else
                emit1(1,symbols'pos(colon1));
              end if;
        when '<' =>
              nextch(ch);
              if ch = '=' then
                emit1(1, symbols'pos(le1));
                nextch(ch);
              else
                emit1(1, symbols'pos(lt1));
             end if;
```

```
        when '>' =>
              nextch(ch);
              if ch = '=' then
                emit1(1, symbols'pos(ge1));
                nextch(ch);
              else
                emit1(1, symbols'pos(gt1));
              end if;
        when '=' =>
              emit1(1, symbols'pos(eq1));
              nextch(ch);
        when '.' =>
              nextch(ch);
              if ch = '.' then
                emit1(1, symbols'pos(dotdot1));
                nextch(ch);
              else
                emit1(1,symbols'pos(dot1));
              end if;
        when others  => error(errors'pos(unknown2)); nextch(ch);
      end case;
    end if;
  end if;
end;

begin  -- mic1
  init;
  getfile;
  cc := 0;
  lc := 0;  ch := ' ';
  while not endfile loop
    nextsym;
  end loop;
  emit1(1,symbols'pos(endtext1));
```

```
  text_io.close(listfile);
  text_io.close(fout1);
  text_io.close(mnfile);
  exception
    when nm_error => null;
end pass1;
```

3. Pass2: Parsing, Scope and Context Analysis.

The syntax given for our language is, as is frequently done in compiling, a 'fake' LL(1) syntax. This means that while the syntax is formally LL(1) so that the syntax analysis works properly the actual language is context sensitive. This is true of virtually all compiler languages which are purported to be context free. In fact such issues as not allowing duplicate declarations of variables or distinguishing between assignment statements or procedure calls while using a one look-ahead parsing algorithm are contextual matters. The literature, however, usually refers to these matters as 'static semantics'. With the exception of type declarations these are generally not semantical issues but a way of restricting to a context sensitive subset of the languages given by our syntax. The restrictions, as we have stated are mostly contextual in nature at this point. The usual way to manage most of these restrictions, including type declarations, is by means of a symbol table. (Sometimes these contextual restrictions are managed in an ad hoc manner, viz. the handling of the dangling else in Pascal.) By use of a symbol table and a set of rules usually given in a language reference manual the contextual restrictions can be imposed in this pass.

For the most part we follow the Ada restrictions but there are a few exceptions. Some of these are

1) We do not allow packages to be nested within packages. For pedagogical reasons this is an unnecessary complication. Moreover we suspect that in practice it is rarely done. One the other hand, since we allow packages to be declared within procedures and procedures within packages we have an indirect way of nesting packages.

2) Parameters to procedures follow the Pascal rules. That is the default is IN or IN may be declared explicitly. This is like the Ada IN, but in our case the variable becomes, as in Pascal, an unrestricted local variable in the procedure. The OUT declaration follows the specification of the Pascal VAR parameter. That is it replaces both the Ada OUT and the Ada IN OUT.

3) Precedence in expressions follow the Pascal rules. These are easy to implement using LL(1) parsing techniques.

4) There is no FOR statement. However the WHILE statement and the LOOP statement with exits have been implemented. Here we follow the philosophy of [H].

5) Records have been introduced but the record variant yields a complication which

which made it necessary to omit it.

6) High dimensional arrays are introduced but only as arrays of arrays. This is reasonably practical.

7) Any typing of an identifier must be done by means of a type identifier which has been previously defined. This simplifies considerably the type analysis. Thus one cannot define

xx: array(0 .. 10) of int;

one must do it in the following way:

type list is array(0 .. 10) of int;

xx: list;

8) The only basic types available in the compiler are integer and boolean. At first it was thought to introduce character types but this was abandoned.

9) Even though this is not really part of the scope analysis it is necessary to point out here that I/O is very primitive.

The main feature of this pass is the symbol table. Here we decided to maintain simplicity and use only one table. That is all identifiers would be on the table as well as all types, procedures and packages. No other tables were introduced. This provided a uniform treatment for identifiers but had the disadvantage that the record entries on the table would be large. However it was felt that space was not a serious consideration. Additionally to avoid complications Ada's variant records were not used in our symbol table. The record structure for the symbol table is shown below.

```
type item is record
        bkwrd, fwrd, lvl: integer;
        id_name: integer;
        kind: idkind;
        typ: typekind;
        trans: modekind;
        typ_point, pk_pt: integer;
        param_end, sect_end: integer;
end record;
```

The symbol table is then declared as

symbol_table: array(0 .. symtop) of item;

The structure of the symbol table is that of a stack, in fact, a multistack. The structure of the language is organized into sections. Sections are started by procedures, records and packages. Each section defines a locality. When the section start is entered on the symbol table the entry sect_end keeps track of the last identifier declared in that section. Each item in the section points back to the previous item of the section and thus the sections can be searched backwards. The two parts of a package both appear on the table and the package body has a pointer, pk_pt which points to the item for the initial package declaration. Thus when the package body is completed the initial package declaration can be searched to see if its declared procedure heads appear in the package body. In addition data declared in the in intial package definition can also be found for use in the body. Types appear as items and an identifier of a given type points back to the type item. Consider the following program:

```
proc aaa is
x, y: bool;
type bb is record
          uu: bool;
          x: int;
        end record;
type ll is array(0 .. 4) of bb;
 proc bbb(u: ll) is
 begin
  null;
 end;
pack ccc is
  proc ddd (x: int; y: out ll);
  proc eee (y: bb);
end pack;

  proc fff is
  kk: ll;
  begin
   ccc.ddd(3, kk);
  end;
```

```
pack body ccc is
  proc ddd (x: int; y: out ll) is
  a: bb;
  begin
    null;
  end;

  proc eee (y: bb) is
  uvw: bool;
  begin
    null;
  end;
vv: int;

  proc ee(y: bb) is
  begin
     null;
  end;
end pack;

begin
 null;
end;
```

A partial view of its symbol table is

inx	name	type	typpt	bkwrd	sect_end	id_name
1	-1	proc3	-1	0	5	
2	33	int3	-1	1	30463	
3	34	bool3	-1	2	7154	
4	37	char3	-1	3	0	
5	57	proc3	30326	4	22	aaa
6	58	bool3	3	5	25186	x

7	59	bool3	3	6	8224	y
8	60	rec3	8	7	10	bb
9	61	bool3	3	8	28192	uu
10	58	int3	2	9	28793	x
11	62	array3	8	8	13344	ll
12	63	proc3	29296	11	13	bbb
13	64	array3	11	12	3438	u
14	65	pkg_def3	8224	12	18	ccc
15	66	proc3	2573	14	17	ddd
16	58	int3	2	15	30063	x
17	59	array3	11	16	8293	y
18	67	proc3	25710	15	19	eee
19	59	rec3	8	18	2573	y
20	68	proc3	8224	14	21	fff
21	69	array3	11	20	27499	kk
22	65	pkg_bdy3	3338	20	31	ccc
23	66	proc3	2573	22	26	ddd
24	58	int3	2	23	30063	x
25	59	array3	11	24	3387	y
26	70	rec3	8	25	27756	a
27	67	proc3	2573	23	29	eee
28	59	rec3	8	27	8224	y
29	71	bool3	3	28	26983	uvw
30	72	int3	2	27	7876	vv
31	73	proc3	2	30	32	ee
32	59	rec3	8	31	-20749	y

The entries with large numbers are uninitialized garbage entries. Notice that the section end of aaa at line 5 is 22. A backward search from 22 ccc (body) yields 20 fff, 14 ccc (decl), 12 bbb, 11 ll, 8 bb, 7 y, 6 x, 5 aaa. Inside this section we find, for example the section starting at line 14 whose section end is 18. Using this multistack arrangement we are able to keep track of all of the localities of the program.

The other main feature of pass2 is of course the parsing. We have chosen panic

parsing as can be found, for example in Turbo Pascal. The program of pass2 follows:

```
with text_io;
package int_io is new text_io.integer_io(integer);
with text_io; with int_io;
procedure pass2 is
 symtop: constant := 500;
 type symbols is (begin1, end1, dot1, elsif1, becomes1, plusop1, --5
                  minusop1,id1, intlit1, lparenth1, rparenth1, --10
                  semicolon1, comma1, endtext1, out1, if1, --15
                  then1, loop1, exit1, lineno1, error1, lt1, le1, --22
                  gt1, ge1, eq1, ne1 , star1, slash1, and1,  --29
                  or1, not1, colon1, int1, bool1,  array1, --35
                  body1, char1, dotdot1, is1, type1, true1, false1,--42
  else1, of1, mod1, null1, pack1, record1, --48
  when1, while1, proc1, in1, write1, writeln1, read1); --55

 type typekind is (int3, bool3, char3, array3, rec3, proc3, pkg_def3,
   pkg_bdy3, notyp3);
 type idkind is (obj4, typ_id4, param4, proc_id4, pkg_id4, unknown_id4);
 int_pt, bool_pt, char_pt: integer;

 type modekind is (in5, out5);

 infil, fout2, listfile, mnfile: text_io.file_type;
 sym: symbols;
 i, linenum, secondsym, errposn : integer;
 endfile: boolean;
 type item is record
                 bkwrd, fwrd, lvl: integer;
                 id_name: integer;
                 kind: idkind;
                 typ: typekind;
                 trans: modekind;
                 typ_point, pk_pt: integer;
```

```
                param_end, sect_end: integer;
              end record;
  symbol_table: array(0 .. symtop) of item;
  double, invisible: array(symbols) of boolean;
  levmax: constant := 25;
  display: array(integer range -1 .. levmax) of integer;
  subtype str11 is string(1 .. 11);
  subtype str20 is string(1 .. 20);
  subtype str8 is string(1 .. 8);
  err_excep: exception;
  table_pt: integer;
  section_start: integer;
  type q_item is record
           idk: idkind;
           pl: integer;
         end record;
  pkg_d_stack: array(1 .. 100) of integer;
  pkg_d_pt: integer;

package debug is
  tmnem: array(typekind) of str8;
  procedure emit(n: integer);
  function length(st: str11) return integer;
  procedure emit1(k,n: integer);
  procedure emit2(m,n:integer);
  procedure error(m,k: integer);
end debug;

package body debug is

  endmntable: constant := 55;
  outlen, mnemlen: integer;
  mnem: array(integer range 0..endmntable) of str11;
```

```
function size(n: integer) return integer is
neg: boolean; nn, len: integer;
begin
  nn := n;
  neg := (nn < 0);
  if neg then nn := -nn; end if;
  if  nn <= 9 then  len := 1;
  elsif nn <= 99 then len := 2;
  elsif nn <= 999 then len := 3;
  elsif nn <= 9999 then len := 4;
  elsif nn <= 32767 then len := 5;
  end if;
  if neg then return len + 1;
  else return len; end if;
end size;

procedure emit(n: integer) is
sz: integer;
begin
  sz := size(n);
  text_io.put(fout2,' '); int_io.put(fout2,n, sz);
  outlen := outlen + sz + 1;
  if outlen >= 72 then
    text_io.put_line(fout2,"");
    outlen := 0;
  end if ;
end emit;

function length(st: str11) return integer is
i: integer;
begin
  i := 11;
  while st(i) = ' ' loop
    i := i - 1;
```

```
 end loop;
 return i;
end;

procedure emit1(k,n: integer) is
lngth, sz: integer;
begin
 if k = 1 then
   lngth := length(mnem(n));
   emit(n);
   text_io.put(mnfile, ' '& mnem(n)(1 .. lngth));
   mnemlen := mnemlen + lngth + 1;
   if mnemlen >= 65 then
     text_io.put_line(mnfile,"");
     mnemlen := 0;
   end if;
 else
    emit(n);
    sz := size(n);
    text_io.put(mnfile,' '); int_io.put(mnfile,n,sz);
    mnemlen := mnemlen + sz + 1;
    if mnemlen >= 65 then
      text_io.put_line(mnfile,"");
      mnemlen := 0;
    end if;
 end if;
end emit1;


procedure emit2(m,n:integer) is
begin
 emit1(1,m); emit1(2,n);
end;
```

```
procedure error(m,k: integer) is
begin
   emit2(symbols'pos(error1), m);
   errposn := m;
   raise err_excep;
end error;

begin --package debug initialization section
  mnem(0) := "begin1    ";
  mnem(1) := "end1      ";
  mnem(2) := "dot1      ";
  mnem(3) := "elsif1    ";
  mnem(4) := "becomes1  ";
  mnem(5) := "plusop1   ";
  mnem(6) := "minusop1  ";
  mnem(7) := "id1       ";
  mnem(8) := "intlit1   ";
  mnem(9) := "lparenth1 ";
  mnem(10) := "rparenth1 ";
  mnem(11) := "semicolon1 ";
  mnem(12) := "comma1    ";
  mnem(13) := "endtext1  ";
  mnem(14) := "out1      ";
  mnem(15) := "if1       ";
  mnem(16) := "then1     ";
  mnem(17) := "loop1     ";
  mnem(18) := "exit1     ";
  mnem(19) := "lineno1   ";
  mnem(20) := "error1    ";
  mnem(21) := "lt1       ";
  mnem(22) := "le2       ";
  mnem(23) := "gt1       ";
  mnem(24) := "ge1       ";
  mnem(25) := "eq1       ";
```

```
mnem(26) := "ne1      ";
mnem(27) := "star1    ";
mnem(28) := "slash1   ";
mnem(29) := "and1     ";
mnem(30) := "or1      ";
mnem(31) := "not1     ";
mnem(32) := "colon1   ";
mnem(33) := "int1     ";
mnem(34) := "bool1    ";
mnem(35) := "array1   ";
mnem(36) := "body1    ";
mnem(37) := "char1    ";
mnem(38) := "dotdot1  ";
mnem(39) := "is1      ";
mnem(40) := "type1    ";
mnem(41) := "true1    ";
mnem(42) := "false1   ";
mnem(43) := "else1    ";
mnem(44) := "of1      ";
mnem(45) := "mod1     ";
mnem(46) := "null1    ";
mnem(47) := "pack1    ";
mnem(48) := "record1  ";
mnem(49) := "when1    ";
mnem(50) := "while1   ";
mnem(51) := "proc1    ";
mnem(52) := "in1      ";
mnem(53) := "write1   ";
mnem(54) := "writeln1 ";
mnem(55) := "read1    ";
outlen := 0; mnemlen := 0;
tmnem := ("int3   ", "bool3  ", "char3  ",
 "array3 ", "rec3   ", "proc3  ", "pkg_def3",
 "pkg_bdy3", "notyp3 ");
```

```
end debug;

package initialize is
  procedure init;
end initialize;

package body initialize is
  procedure init is
  begin
    for ss in symbols loop
      double(ss) := false;
      invisible(ss) := false;
    end loop;
    double(id1) := true;
    double(intlit1) := true;
    double(lineno1) := true;
    double(error1) := true;
    invisible(lineno1) := true;
    invisible(error1) := true;
    pkg_d_pt := 0;
  end init;
begin
  init;
end initialize;

procedure  getfile is
 use text_io;
begin
  open(infil,in_file,"out1");
  create(listfile,out_file,"lister");
  create(fout2,out_file,"out2");
  -- The file mnem2 is used for debugging purposes.
  -- It can be eliminated later.
  create(mnfile, out_file, "mnem2");
```

```
  text_io.put_line(mnfile,"pass2.ada mnem2");
  text_io.new_line(mnfile);
end;


procedure nextsym is
firstsym: integer;
use int_io;
begin
  get(infil,firstsym);
  sym := symbols'val(firstsym);
  while invisible(sym) loop
    debug.emit1(1,firstsym);
    get(infil, secondsym);
    debug.emit1(2, secondsym);
    if sym = lineno1 then
      linenum := secondsym;
    elsif sym = error1 then
      debug.error(1, secondsym);
    end if;
    get(infil, firstsym);
    sym := symbols'val(firstsym);
  end loop;
  debug.emit1(1,firstsym);
  if double(sym) then
    get(infil, secondsym);
    debug.emit1(2, secondsym);
  end if;
end nextsym;

package b_entry is
procedure basic_entry(id,tpt: integer;
                      knd:idkind; tp:typekind);
end b_entry;
```

```
package body b_entry is
procedure basic_entry(id,tpt: integer;
                      knd:idkind; tp:typekind) is
begin
   table_pt := table_pt + 1;
   symbol_table(table_pt).bkwrd := table_pt - 1;
   symbol_table(table_pt).id_name := id;
   symbol_table(table_pt).lvl := -1;
   symbol_table(table_pt).kind := knd;
   symbol_table(table_pt).typ := tp;
   symbol_table(table_pt).typ_point := tpt;
end;

begin -- b_entry
  table_pt := 0; display(-1) := 1;
  basic_entry(-1, -1, proc_id4, proc3);
  basic_entry(symbols'pos(int1), -1, typ_id4, int3); int_pt := table_pt;
  basic_entry(symbols'pos(bool1), -1, typ_id4, bool3); bool_pt := table_pt;
  basic_entry(symbols'pos(char1), -1, typ_id4, char3); char_pt := table_pt;
  symbol_table(1).sect_end := 4;
end b_entry;

procedure checksym(symb: symbols; n: integer) is
begin
   if sym = symb then nextsym;
   else debug.error(n,symbols'pos(sym)); end if;
end;

procedure checksemi(n: integer) is
begin
  checksym(semicolon1,n);
end;
```

```
procedure show_table is
begin
  text_io.put_line(" inx    name    type  typpt   bwrd   sctnd");
  for i in 1 .. table_pt loop
    int_io.put(i,3);
    int_io.put(symbol_table(i).id_name,9);
    text_io.put("   ");
    text_io.put(debug.tmnem(symbol_table(i).typ));
    int_io.put(symbol_table(i).typ_point,7);
    int_io.put(symbol_table(i).bkwrd,7);
    int_io.put(symbol_table(i).sect_end,7);
    text_io.new_line;
  end loop;
end;

procedure enter_var(iden, lv: integer; knd: idkind) is
j, k: integer;
begin
  table_pt := table_pt + 1;
  symbol_table(table_pt).bkwrd :=
                  symbol_table(display(lv)).sect_end;
  symbol_table(table_pt).id_name := iden;
  symbol_table(table_pt).lvl := lv;
  symbol_table(table_pt).kind := knd;
  symbol_table(table_pt).typ := notyp3;
  symbol_table(display(lv)).sect_end := table_pt;
  symbol_table(table_pt).pk_pt := -1;
end;

procedure one_sect_check(ss,tp: integer; pl: out integer) is
temp, j: integer;
-- checks a section for an id
begin
  temp := symbol_table(tp).bkwrd;  -- New code
```

```
  symbol_table(tp).bkwrd := 0;  -- New code
  symbol_table(0).id_name := ss;
  j := symbol_table(tp).sect_end;
  while symbol_table(j).id_name /= ss loop
    j := symbol_table(j).bkwrd;
  end loop;
  symbol_table(tp).bkwrd := temp;
  pl := j;
end;


procedure check_new_id(n, ss, lvl: integer; pl : out integer) is
temp, j, k, pp, tp: integer;
begin
  tp := display(lvl);
  symbol_table(0).id_name := ss;
  one_sect_check(ss,tp,j);
-- If the section is a pack body then special conditions prevail.
-- For object variables and type variables check_new_id should consider
-- the pack def as part of the search domain. In the case of procedures
-- however check_new_id should only search within the pack body. A second
-- check later will verify that the pack def procedures have a body.
 if n = 1 then
  if (j = 0) and (symbol_table(tp).typ = pkg_bdy3) then
     pp := symbol_table(display(lvl)).pk_pt;
     one_sect_check(ss, pp, k);
  end if;
 end if;
  pl := j;
end;

function id_index(id, lvl: integer) return integer is
lv, lnk, temp, tp, pp: integer;
begin
```

```
    lv := lvl; symbol_table(0).id_name := id;
    loop
      tp := display(lv);
      one_sect_check(id,tp,lnk);
      if (lnk = 0) and (symbol_table(tp).typ = pkg_bdy3) then
        pp := symbol_table(tp).pk_pt;
        one_sect_check(id,pp,lnk);
       end if;
      lv := lv - 1;
      exit when (lv < - 1) or (lnk /= 0);
    end loop;
    return lnk;
  end;

  procedure id_list_tail(lvl: integer) is
  place: integer;
  begin
    if sym = comma1 then
      nextsym;
      if sym = id1 then
        check_new_id(1, secondsym,lvl, place);
        if place = 0 then enter_var(secondsym,lvl,obj4);
        else debug.error(7,symbols'pos(id1)); end if;
        nextsym;
      else debug.error(8,symbols'pos(id1)); end if;
      id_list_tail(lvl);
    else
      -- test follow symbols for id_list_tail
      if sym /= colon1 then debug.error(6,symbols'pos(colon1));
      end if;
    end if;
  end;

  procedure id_list(lvl: integer) is
```

```
place : integer;
begin
 if sym = id1 then
   check_new_id(1, secondsym, lvl, place);
   if place = 0 then enter_var(secondsym,lvl,obj4);
   else debug.error(5,symbols'pos(id1)); end if;
   nextsym;
 else debug.error(6,symbols'pos(id1)); end if;
 id_list_tail(lvl);
end;

procedure type_ind(lvl: integer; tp: out integer) is
j: integer;
begin
  case sym is
    when id1 =>
     j:= id_index(secondsym, lvl);
     if j = 0 then debug.error(9,symbols'pos(id1));
     elsif symbol_table(j).kind /= typ_id4 then
       debug.error(10,symbols'pos(id1));
     else tp := j;
     end if;
     nextsym;
    when int1 =>
         tp := 2; nextsym;
    when bool1 =>
         tp := 3; nextsym;
    when char1 =>
         tp := 4; nextsym;
    when others =>
     debug.error(11,symbols'pos(id1));
  end case;
end;
```

```
procedure fill_type(st, fin, tp: integer) is
tt: typekind;
begin
  tt := symbol_table(tp).typ;
  for i in st .. fin loop
    symbol_table(i).typ_point := tp;
    symbol_table(i).typ := tt;
  end loop;
end;

procedure object_decl(lvl: integer) is
strt_pt, end_pt, tp_pt: integer;
begin
   strt_pt := table_pt + 1;
   id_list(lvl);
   end_pt := table_pt;
   checksym(colon1, 3);
   type_ind(lvl,tp_pt);
   fill_type(strt_pt, end_pt, tp_pt);
   checksemi(4);
end;

procedure stype_def(lvl: integer) is
se: integer;
begin
   se := symbol_table(display(lvl)).sect_end;
   case sym is
     when int1 =>  symbol_table(se).typ_point := int_pt;
                   symbol_table(se).typ := int3;
     when bool1 =>  symbol_table(se).typ_point := bool_pt;
                   symbol_table(se).typ := bool3;
     when char1 =>  symbol_table(se).typ_point := char_pt;
                   symbol_table(se).typ := char3;
```

```
    when others => debug.error(17, symbols'pos(type1));
  end case;
  nextsym; -- this placement of nextsym would not be
          -- correct if this were not a panic parser.
          -- Where should it go?
end;

procedure index_bd(val: out integer) is
begin
  case sym is
    when intlit1 =>
      val := secondsym:
      nextsym;
    when plusop1 =>
      nextsym;
      if sym = intlit1 then
        val := secondsym;
        nextsym;
      else
        debug.error(230, symbols'pos(intlit1));
      end if;
    when minusop1 =>
      nextsym;
      if sym = intlit1 then
        val := - secondsym;
        nextsym;
      else
        debug.error(230, symbols'pos(intlit1));
      end if;
    when others =>
      debug.error(231,symbols'pos(intlit1));
  end case;
end;
```

```
procedure index_range is
low, high: integer;
begin
   index_bd(low);
   checksym(dotdot1, 23);
   index_bd(high);
   if high < low then debug.error(26, symbols'pos(array1));
   end if;
end;

procedure array_def(lvl: integer) is
tp, j: integer;
begin
  checksym(array1, 18);
  checksym(lparenth1, 19);
  index_range;
  checksym(rparenth1,  21);
  checksym(of1, 20);
  type_ind(lvl, tp);
  j := symbol_table(display(lvl)). sect_end;
  symbol_table(j).typ_point := tp;
  symbol_table(j).typ := array3;
end;

procedure comp_def_tail(lvl: integer) is
begin
   if sym = id1 then  -- is the first symbol of object_decl
     object_decl(lvl);
     comp_def_tail(lvl);
   elsif sym /= end1 then  -- takes care of follow symbols
    debug.error(29,symbols'pos(id1));
   end if;
end;
```

```
procedure comp_def(lvl: integer) is
begin
  object_decl(lvl);
  comp_def_tail(lvl);
end;

procedure record_def(lvl: integer) is
temp: integer; tp: integer;
begin
  checksym(record1, 25);
  symbol_table(table_pt).typ := rec3;
  tp := table_pt;
  symbol_table(tp).sect_end := tp;
  symbol_table(tp).typ_point := tp;
  display(lvl+1) := tp;
  comp_def(lvl+1);
  checksym(end1, 27);
  checksym(record1, 28);
  symbol_table(tp).sect_end := table_pt;
end;

procedure type_def(lvl: integer) is
begin
 case sym is
  when int1 | bool1 | char1 => stype_def(lvl);
  when array1 => array_def(lvl);
  when record1 => record_def(lvl);
  when others => debug.error(16,symbols'pos(type1));
 end case;
end;

procedure type_decl(lvl: integer) is
pl: integer;
begin
```

```
 checksym(type1,12);
 if sym = id1 then
  check_new_id(1, secondsym,lvl, pl);
  if pl /= 0 then
   debug.error(13,symbols'pos(id1));
  else
   enter_var(secondsym, lvl, typ_id4);
  end if;
  nextsym;
 else
  debug.error(14,symbols'pos(id1));
 end if;
 checksym(is1,14);
 type_def(lvl);
 checksemi(15);
end;

procedure mode(md: out modekind) is
begin
  case sym is
   when in1 => md := in5; nextsym;
   when out1 => md := out5; nextsym;
   -- deal with follow symbols
   when id1 | int1 | bool1 | char1 => md := in5;
   when others => debug.error(46,symbols'pos(type1));
  end case;
end;

procedure p_tail(lvl: integer) is
place: integer;
begin
 if sym = comma1 then
    checksym(comma1,40);
    if sym = id1 then
```

```
        check_new_id(2, secondsym, lvl, place);
        if place = 0 then enter_var(secondsym,lvl,param4);
        else debug.error(41,symbols'pos(id1)); end if;
nextsym;
p_tail(lvl);
    else debug.error(42,symbols'pos(id1)); end if;
  elsif sym /= colon1 then
    debug.error(43, symbols'pos(id1));
  end if;
end;

procedure sparam_decl(lvl: integer) is
strt_pt, end_pt, tp, i: integer; tt:typekind;
md: modekind; place: integer;
begin
  if sym = id1 then
    check_new_id(2, secondsym, lvl, place);
    if place = 0 then
      strt_pt := table_pt + 1;
      enter_var(secondsym,lvl,param4);
    else debug.error(37,symbols'pos(id1)); end if;
    nextsym;
  else debug.error(38,symbols'pos(id1)); end if;
  p_tail(lvl);
  end_pt := table_pt;
  checksym(colon1,39);
  mode(md);
  type_ind(lvl, tp);
  tt := symbol_table(tp).typ;
  for i in strt_pt .. end_pt loop
    symbol_table(i).typ_point := tp;
    symbol_table(i).typ := tt;
    symbol_table(i).trans := md;
  end loop;
```

```
  -- semicolon check is done in p_dec_tail
end;

procedure p_dec_tail(lvl: integer) is
begin
  if sym = semicolon1 then
      checksemi(44);
      sparam_decl(lvl);
      p_dec_tail(lvl);
  elsif sym /= rparenth1 then
    debug.error(45, symbols'pos(rparenth1));
  end if;
end;

procedure param_decl(lvl: integer) is
begin
  sparam_decl(lvl);
  p_dec_tail(lvl);
end;

procedure formal_part(lvl: integer) is
begin
 if sym = lparenth1 then
    checksym(lparenth1,35);
    param_decl(lvl);
    checksym(rparenth1.36);
 elsif sym /= semicolon1 and sym /= is1 then
    debug.error(37,symbols'pos(proc1));
 end if;
end;

procedure subprog_header(lvl: integer) is
pl, tp: integer;
begin
```

```
  checksym(proc1,32);
  if sym = id1 then
    check_new_id(2, secondsym,lvl, pl);
    if pl /= 0 then
       debug.error(33,symbols'pos(id1));
    else
      tp := table_pt + 1;
      enter_var(secondsym, lvl, proc_id4);
      symbol_table(tp).sect_end := tp;
      symbol_table(tp).param_end := tp;
      symbol_table(tp).typ := proc3;
    end if;
    nextsym;
  else
    debug.error(34,symbols'pos(id1));
  end if;
  display(lvl+1) := tp;
  formal_part(lvl+1);
  symbol_table(tp).param_end :=
    symbol_table(tp).sect_end;
end;

procedure expression(lvl: integer; typ_pt: out integer);

procedure vbl(lvl: integer; typ_pt: out integer);

procedure factor(lvl: integer; typ_pt: out integer) is
tpt: integer;
begin
  case sym is
    when intlit1 =>
typ_pt := int_pt;
nextsym;
    when lparenth1 =>
```

```
nextsym;
expression(lvl, typ_pt);
checksym(rparenth1, 160);
    when id1 => vbl(lvl, typ_pt);
    when not1 =>
nextsym;
factor(lvl, tpt);
if tpt /= bool_pt then
 debug.error(161, symbols'pos(not1));
else
 typ_pt := bool_pt;
end if;
    when true1 | false1 =>
nextsym;
typ_pt := bool_pt;
    when others => debug.error(162,symbols'pos(id1));
  end case;
end;

procedure term_tail(lvl, typ_pt: integer) is
tpt1, tpt2: integer; arith_op: boolean;
begin
   case sym is
    when and1 | star1 | mod1 | slash1 =>
if sym = and1 then
 tpt1 := bool_pt;
else
 tpt1 := int_pt;
end if;
if typ_pt /= tpt1 then
 debug.error(163, symbols'pos(and1));
else
 nextsym;
 factor(lvl, tpt2);
```

```
  if tpt2 /= typ_pt then
    debug.error(164, symbols'pos(and1));
  end if;
  term_tail(lvl,typ_pt);
end if;
      when colon1 | semicolon1 | rparenth1 | comma1 | eq1 | ne1 |
   lt1 | le1 | gt1 | ge1 | plusop1 | minusop1 | or1 |
   then1 | loop1 => null;
      when others => debug.error(165, symbols'pos(and1));
    end case;
end;

procedure term(lvl: integer; typ_pt: out integer) is
tpt: integer;
begin
   factor(lvl, tpt);
   term_tail(lvl,tpt);
   typ_pt := tpt;
end;

procedure sexp_tail(lvl, typ_pt: integer) is
tpt1, tpt2: integer;
begin
   case sym is
     when plusop1 | minusop1 | or1 =>
if sym = or1 then
  tpt1 := bool_pt;
else
  tpt1 := int_pt;
end if;
if typ_pt /= tpt1 then
  debug.error(166, symbols'pos(or1));
else
   nextsym;
```

```
    term(lvl, tpt2);
    if typ_pt /= tpt2 then
      debug.error(167,symbols'pos(or1));
    end if;
    sexp_tail(lvl, tpt2);
  end if;
      when colon1 | semicolon1 | rparenth1 | comma1 | eq1 | ne1 |
    lt1 | le1 | gt1 | ge1 | then1 | loop1 => null;
      when others =>
  debug.error(168, symbols'pos(or1));
    end case;
  end;

  procedure simp_expr(lvl: integer; typ_pt: out integer) is
  tpt1, tpt2: integer; aflag: boolean;
  begin
    case sym is
      when plusop1 | minusop1 =>
        aflag := true;
        nextsym;
      when others =>
        aflag := false;
    end case;
    term(lvl, tpt1);
    if aflag and (tpt1 /= int_pt) then
      debug.error(169, symbols'pos(plusop1));
    end if;
    sexp_tail(lvl,tpt1);
    typ_pt := tpt1;
  end;

  procedure exp_tail(lvl: integer; typ_pt: in out integer) is
  tpt: integer; bflag: boolean;
  begin
```

```
    case sym is
when eql | nel | ltl | lel | gtl | gel  =>
  nextsym;
  simp_expr(lvl, tpt);
  if typ_pt /= tpt then
    debug.error(171,symbols'pos(eql));
  elsif (tpt /= int_pt) and (tpt /= bool_pt) then
    debug.error(300,symbols'pos(eql));
  else
    typ_pt := bool_pt;
  end if;
when colonl | semicolonl | rparenthl | commal | thenl | loopl =>
  null;
when others =>
  debug.error(172,symbols'pos(eql));
    end case;
end;

procedure expression(lvl: integer; typ_pt: out integer) is
tpt1, tpt2: integer;
begin
  simp_expr(lvl, tpt1);
  tpt2 := tpt1;
  exp_tail(lvl, tpt2);
  typ_pt := tpt2;
end;

procedure vbl_tail(lvl:integer; typ_pt: in out integer) is
tpt: integer;
begin
   case sym is
    when lparenthl =>
  checksym(lparenthl, 120);
  expression(lvl, tpt);
```

```
 if tpt /= int_pt then
  debug.error(121,symbols'pos(id1));
 end if;
 checksym(rparenth1, 140);
 typ_pt := symbol_table(typ_pt).typ_point;
 vbl_tail(lvl,typ_pt);
    when  colon1 | semicolon1 | rparenth1 | comma1 | eq1 | ne1 |
  lt1 | le1 | gt1 | ge1 | plusop1 | minusop1 | and1 |
  star1 | mod1 | slash1 | or1 | then1 | loop1 | dot1 | becomes1
  => null; -- typ_pt remains unchanged
    when others =>
  debug.error(122, symbols'pos(id1));
   end case;
end;

procedure vbl1(lvl, pl: integer; typ_pt: out integer);

procedure v1_tail(lvl, pl, pl0: integer; typ_pt :out integer) is
pl1, pl2: integer;
begin
 case sym is
  when dot1 =>
     checksym(dot1,117);
     if sym /= id1 then
 debug.error(119, symbols'pos(id1));
     else
 pl1 := symbol_table(pl).typ_point;
 one_sect_check(secondsym,pl1,pl2);
 vbl1(lvl,pl2,typ_pt);
     end if;
  when colon1 | semicolon1 | rparenth1 | comma1 | eq1 | ne1 | lt1 | le1 |
gt1 | ge1 | plusop1 | minusop1 | and1 | star1 | mod1 |
slash1 | or1 | then1 | loop1 | becomes1  => typ_pt := pl0;
  when others =>  debug.error(205, symbols'pos(id1));
```

```
 end case;
end;

procedure vbl1(lvl, pl: integer; typ_pt: out integer) is
dummy, tpt1, tpt2: integer;
begin
     dummy := 0;
     if sym /= id1 then
debug.error(113, symbols'pos(id1));
     else
nextsym;
case symbol_table(pl).typ is
  when rec3 =>
      tpt1 := symbol_table(pl).typ_point;
      v1_tail(lvl, pl, tpt1, tpt2);
      typ_pt := tpt2;
  when int3 =>
      vbl_tail(lvl, dummy);
      typ_pt := int_pt;
  when bool3 =>
      vbl_tail(lvl, dummy);
      typ_pt := bool_pt;
  when array3 =>
      tpt1 := symbol_table(pl).typ_point;
      vbl_tail(lvl,tpt1);
      v1_tail(lvl, tpt1, tpt1, tpt2);
      typ_pt := tpt2;
  when others => debug.error(141,symbols'pos(id1));
 end case;
      end if;
end;

procedure vbl2(lvl, pl: integer; typ_pt: out integer) is
pl1: integer;
```

```
begin
  if sym /= id1 then
    debug.error(125,symbols'pos(id1));
  elsif symbol_table(pl).kind /= pkg_id4 then
    debug.error(126,symbols'pos(id1));
  else
    nextsym;
    checksym(dot1, 127);
    if sym /= id1 then
      debug.error(128, symbols'pos(id1));
    else
      one_sect_check(secondsym,pl,pl1);
      vbl1(lvl, pl1, typ_pt);
    end if;
  end if;
end;

procedure vbl(lvl: integer; typ_pt: out integer) is
pl, pl1: integer;
begin
  if sym /= id1 then
    debug.error(130, symbols'pos(id1));
  else
    pl := id_index(secondsym, lvl);
    case symbol_table(pl).kind is
      when obj4 | param4 => vbl1(lvl, pl, typ_pt);
      when pkg_id4 =>
 pl1 := symbol_table(pl).pk_pt;
 vbl2(lvl, pl1, typ_pt);
      when others => debug.error(131, symbols'pos(id1));
    end case;
  end if;
end;
```

```
procedure exit_stat(lvl: integer) is
tpt: integer;
begin
   checksym(exit1,103);
   checksym(when1,104);
   expression(lvl,tpt);
   if symbol_table(tpt).typ /= bool3 then
    debug.error(105,symbols'pos(bool1));
   end if;
end;

procedure p_sub_tail(lvl, pl, pl0: integer);

procedure exp_p_sub(lvl, pl, pl1: integer) is
tpt1, pl2: integer;
begin
  if symbol_table(pl1).kind /= param4 then
    debug.error(190,symbols'pos(id1));
  elsif symbol_table(pl1).trans = in5 then
    expression(lvl, tpt1);
    if symbol_table(pl1).typ_point /= tpt1 then
      debug.error(191, symbols'pos(id1));
    else
p_sub_tail(lvl, pl, pl1);
    end if;
  elsif symbol_table(pl1).trans = out5 then  --redundant check
     vbl(lvl, tpt1);
     if symbol_table(pl1).typ_point /= tpt1 then
 debug.error(195, symbols'pos(id1));
     else
p_sub_tail(lvl, pl, pl1);
     end if;
  end if;
end;
```

```
procedure p_sub_tail(lvl, pl, pl0: integer) is
tpt1, pl1, pl2: integer;
begin
 case sym is
   when comma1 =>
nextsym;
pl1 := pl0 + 1;
if symbol_table(pl).param_end < pl1 then
  debug.error(189, symbols'pos(id1));
else
 exp_p_sub(lvl, pl, pl1);
end if;
   when rparenth1 =>
     if pl0 /= symbol_table(pl).param_end then
  debug.error(198, symbols'pos(id1));
     end if;
   when others => debug.error(197, symbols'pos(id1));
 end case;
end;

procedure p_call_tail(lvl, pl: integer) is
tpt1, pl1, pl2: integer;
begin
 case sym is
   when lparenth1 =>
     if symbol_table(pl).param_end <= pl then
debug.error(179,symbols'pos(id1));
     else
nextsym;
pl1 := pl + 1;
exp_p_sub(lvl,pl,pl1);
checksym(rparenth1, 182);
     end if;
```

```
    when semicolon1 =>
      if pl /= symbol_table(pl).param_end then
  debug.error(188, symbols'pos(id1));
      end if;
    when others => debug.error(187, symbols'pos(id1));
  end case;
end;

procedure proc_call_stat(lvl, pl: integer) is
begin
--pl points to symbol table entyr for id
--which is a proc id
    nextsym;
    p_call_tail(lvl, pl);
end;

procedure assign_stat(lvl, pl: integer) is
typ1, typ2: integer;
begin
    vbl1(lvl, pl, typ1);
    checksym(becomes1, 97);
    expression(lvl,typ2);
    if typ1 /= typ2 then
      debug.error(98,symbols'pos(id1));
    end if;
end;

procedure pk_stat_tail(lvl, pl: integer) is
pl1: integer;
begin
-- now seeking id in package located at pl
  if sym = id1 then
    one_sect_check(secondsym,pl,pl1);
    if pl1 /= 0 then
```

```
      case symbol_table(pl1).kind is
        when proc_id4 => proc_call_stat(lvl, pl1);
when obj4 | param4 => assign_stat(lvl, pl1);
        when others => debug.error(94, symbols'pos(pack1));
      end case;
    else
      debug.error(95,symbols'pos(pack1));
    end if;
  else
    debug.error(96,symbols'pos(pack1));
  end if;
end;


procedure pk_stat(lvl, pl: integer) is
begin
-- pl is the address of the pack def
   if sym = id1 then
      nextsym;
   else
      debug.error(115, symbols'pos(id1));
   end if;
   checksym(dot1,93);
   pk_stat_tail(lvl, pl);
end;

procedure read_stat(lvl: integer) is
tpt: integer;
begin
   nextsym;
   checksym(lparenth1, 306);
   vbl(lvl, tpt);
   if (tpt /= int_pt) and (tpt /= bool_pt) then
      debug.error(309, symbols'pos(id1));
```

```
  end if;
  checksym(rparenth1, 320);
end;

procedure w_tail(lvl: integer) is
tpt: integer;
begin
  case sym is
    when colon1 => nextsym;
expression(lvl, tpt);
if tpt /= int_pt then
 debug.error(305, symbols'pos(id1));
end if;
    when rparenth1 => null;
    when others => debug.error(304, symbols'pos(id1));
  end case;
end;

procedure write_body(lvl: integer) is
tpt: integer;
begin
  checksym(lparenth1, 301);
  expression(lvl, tpt);
  if (tpt /= int_pt) and (tpt /= bool_pt) then
    debug.error(302, symbols'pos(id1));
  end if;
  w_tail(lvl);
  checksym(rparenth1,303);
end;

procedure write_stat(lvl: integer) is
begin
  case sym is
    when write1 =>
```

```
      nextsym;
      write_body(lvl);
    when writeln1 =>
      nextsym;
    wnen others => null;
  end case;
end;

procedure simp_stat(lvl: integer) is
pl, pl1, pl2: integer;
begin
  case sym is
    when null1 => checksym(null1,901);
    when id1 =>
       pl := id_index(secondsym,lvl);
       case symbol_table(pl).kind is
         when proc_id4 => proc_call_stat(lvl, pl);
  when obj4 | param4 => assign_stat(lvl, pl);
  when pkg_id4 =>
pl := symbol_table(pl).pk_pt; -- make sure it points to
     -- pack def
pk_stat(lvl,pl);
  when others => debug.error(91,symbols'pos(id1));
       end case;
    when semicolon1 => null;
    when write1 | writeln1 => write_stat(lvl);
    when read1 => read_stat(lvl);
    when others => debug.error(92,symbols'pos(id1));
   end case;
end;

procedure stat_seq(lvl: integer);

procedure sing_elsif_part(lvl: integer) is
```

```
tpt: integer;
begin
     checksym(elsif1, 331);
     expression(lvl, tpt);
     if tpt /= bool_pt then
debug.error(151,symbols'pos(elsif1));
     end if;
     checksym(then1,152):
     stat_seq(lvl);
end;

procedure elsif_part(lvl: integer) is
begin
  case sym is
    when elsif1 =>
 sing_elsif_part(lvl);
 elsif_part(lvl);
    when end1 | else1 => null;
    when others =>
debug.error(330, symbols'pos(elsif1));
  end case;
end;

procedure else_part(lvl: integer) is
begin
  case sym is
    when else1 =>
nextsym;
stat_seq(lvl);
    when end1 => null;
    when others => debug.error(154, symbols'pos(else1));
  end case;
end;
```

```
procedure if_stat(lvl: integer) is
tpt: integer;
begin
  checksym(if1,145);
  expression(lvl, tpt);
  if tpt /= bool_pt then
   debug.error(146, symbols'pos(if1));
  end if;
  checksym(then1, 1451);
  stat_seq(lvl);
  elsif_part(lvl);
  else_part(lvl);
  checksym(end1,147);
  checksym(if1,148);
end;

procedure lstat_seq(lvl: integer);

procedure loop_stat(lvl: integer) is
begin
  checksym(loop1,133);
  lstat_seq(lvl);
  checksym(end1,134);
  checksym(loop1,135);
end;

procedure while_stat(lvl: integer) is
tpt: integer;
begin
  checksym(while1,137);
  expression(lvl, tpt);
  if tpt /= bool_pt then
    debug.error(138, symbols'pos(bool1));
  end if;
```

```
 loop_stat(lvl);
end;

procedure comp_stat(lvl: integer) is
begin
  case sym is
   when if1 => if_stat(lvl);
   when loop1 => loop_stat(lvl);
   when while1 => while_stat(lvl);
   when others => debug.error(106, symbols'pos(id1));
  end case;
end;

procedure statement(lvl: integer) is
begin
  case sym is
    when id1 | null1 | write1 | writeln1 | read1 =>
 simp_stat(lvl); checksemi(200);
    when if1 | loop1 | while1 => comp_stat(lvl); checksemi(201);
    when exit1 => debug.error(400, symbols'pos(exit1));
    when others => debug.error(90, symbols'pos(if1));
  end case;
end;

procedure lstatement(lvl: integer) is
begin
  case sym is
    when id1 | null1 | write1 | writeln1 | read1 |
  if1 | loop1 | while1 => statement(lvl);
    when exit1 => exit_stat(lvl); checksemi(401);
    when others => debug.error(402, symbols'pos(if1));
  end case;
end;
```

```
procedure stat_seq_tail(lvl: integer) is
begin
  case sym is
   when id1 | null1 | if1 | loop1 | while1 |
write1 | writeln1 | read1 =>
       statement(lvl);
       stat_seq_tail(lvl);
   when end1 | elsif1 | else1 => null;
   when exit1 => debug.error(403, symbols'pos(exit1));
   when others =>
     debug.error(85, symbols'pos(id1));
  end case;
end;

procedure lstat_seq_tail(lvl: integer) is
begin
  case sym is
   when id1 | null1 | if1 | exit1 | loop1 | while1 |
write1 | writeln1 | read1 =>
       lstatement(lvl);
       lstat_seq_tail(lvl);
   when end1  => null;
   when others =>
     debug.error(411, symbols'pos(id1));
  end case;
end;

procedure stat_seq(lvl: integer) is
begin
  statement(lvl);
  stat_seq_tail(lvl);
end;

procedure lstat_seq(lvl: integer) is
```

```
begin
  lstatement(lvl);
  lstat_seq_tail(lvl);
end;

procedure decl_part(lvl: integer);

procedure subprog_part(lvl: integer) is
begin
   pkg_d_pt := pkg_d_pt + 1;
   pkg_d_stack(pkg_d_pt) := 0;
   decl_part(lvl);
   if pkg_d_stack(pkg_d_pt) /= 0 then
     debug.error(112, symbols'pos(pack1));
   else
     pkg_d_pt := pkg_d_pt - 1;
   end if;
   checksym(begin1,50);
   stat_seq(lvl);
   checksym(end1, 51);
end;

procedure subprog_decl(lvl: integer) is
tp, backlnk: integer;
begin
  subprog_header(lvl);
  checksym(is1, 30);
  subprog_part(lvl + 1);
  checksemi(31);
end;

procedure pkg_d_decl(lvl: integer) is
begin
    case sym is
```

```
      when id1 => object_decl(lvl); pkg_d_decl(lvl);
      when type1 => type_decl(lvl); pkg_d_decl(lvl);
      when proc1 => subprog_header(lvl); checksemi(64);
                    pkg_d_decl(lvl);
      when end1 => null;
      when others => debug.error(75, symbols'pos(pack1));
    end case;
end;

procedure pkg_def_decl(lvl: integer) is
begin
  pkg_d_stack(pkg_d_pt) := pkg_d_stack(pkg_d_pt) + 1;
  pkg_d_decl(lvl);
  checksym(end1,61);
  checksym(pack1,62);
  checksemi(63);
end;

procedure pkg_b_decl(lvl:integer) is
begin
    case sym is
      when id1 => object_decl(lvl); pkg_b_decl(lvl);
      when type1 => type_decl(lvl); pkg_b_decl(lvl);
      when proc1 => subprog_decl(lvl);
                    pkg_b_decl(lvl);
      when end1 => null;
      when others => debug.error(74, symbols'pos(pack1));
    end case;
end;

procedure pkg_body_decl(lvl: integer) is
begin
  pkg_d_stack(pkg_d_pt) := pkg_d_stack(pkg_d_pt) - 1;
  pkg_b_decl(lvl);
```

```
  checksym(end1,70);
  checksym(pack1,71);
  checksemi(72);
end;

procedure package_decl(lvl: integer);

function proc_check(psn1, psn2: integer) return boolean is
j, sz1, sz2: integer; ok: boolean;
-- this procedure checks to see that the proc header
-- declarations in pack def and pack body are the same
begin
  if symbol_table(psn2).kind /= proc_id4 then
    debug.error(80, symbols'pos(pack1));
    return(false);
  elsif symbol_table(psn1).id_name /=
     symbol_table(psn2).id_name  then
    debug.error(77, symbols'pos(pack1));
    return(false);
  else
      sz1 := symbol_table(psn1).param_end - psn1;
      sz2 := symbol_table(psn2).param_end - psn2;
      if sz1 /= sz2 then
        debug.error(78, symbols'pos(pack1));
        return(false);
      elsif sz1 /= 0 then
        j := 1;
        ok := true;
        loop
          ok := symbol_table(psn1+j).id_name =
                  symbol_table(psn2+j).id_name;
          if ok then ok := symbol_table(psn1+j).kind = param4;
          end if;
          if ok then ok := symbol_table(psn2+j).kind = param4;
```

```
          end if;
          if ok then ok := symbol_table(psn1+j).typ_point =
                        symbol_table(psn2+j).typ_point;
          end if;
          if ok then ok := symbol_table(psn1+j).trans =
                        symbol_table(psn2+j).trans;
          end if;
          if not ok then debug.error(79,symbols'pos(pack1));
          end if;
          if ok then
           j := j + 1;
           ok := j <= sz1;
          end if;
          exit when not ok;
        end loop;
      end if;
      return(j > sz1);
  end if;
end;

procedure pkg_search(pp,tp: integer) is
pt1, pt2, pt3, nm, temp: integer; ok: boolean;
begin
-- this procedure checks to see that the proc headers in pack def
-- appear in pack body for the same pack id.
-- pp points to pack def, tp points to pack body
   pt1 := symbol_table(pp).sect_end;
   pt2 := symbol_table(tp).sect_end;
   temp := symbol_table(tp).bkwrd;
   symbol_table(tp).bkwrd := 0;
   while pt1 /= pp loop
     if symbol_table(pt1).kind = proc_id4 then
       nm := symbol_table(pt1).id_name;
       pt3 := pt2;
```

```
        symbol_table(0).id_name := nm;
        while symbol_table(pt3).id_name /= nm loop
          pt3 := symbol_table(pt3).bkwrd;
        end loop;
        if pt3 = 0 then
          debug.error(110,symbols'pos(proc1));
        elsif not proc_check(pt1,pt3) then
          debug.error(111,symbols'pos(proc1));
        end if;
      end if;
      pt1 := symbol_table(pt1).bkwrd;
    end loop;
    symbol_table(tp).bkwrd := temp;
end;

procedure pkg_tail(lvl: integer) is
pl, j, tp, pp: integer;
begin
     case sym is
       when id1 =>
          check_new_id(2, secondsym, lvl, pl);
  if pl = 0 then
     enter_var(secondsym, lvl, pkg_id4);
     symbol_table(table_pt).typ := pkg_def3;
           display(lvl+1) := table_pt;
           symbol_table(table_pt).sect_end := table_pt;
     symbol_table(table_pt).pk_pt := table_pt;
  else debug.error(53,symbols'pos(pack1)); end if;
          nextsym;
          checksym(is1,54);
  pkg_def_decl(lvl+1);
       when body1 =>
          nextsym;
          if sym = id1 then
```

```
          check_new_id(2, secondsym, lvl, pl);
          if pl > 0 then
           if symbol_table(pl).typ /= pkg_def3 then
            debug.error(59, symbols'pos(pack1));
           else
            enter_var(secondsym, lvl, pkg_id4);
            tp := table_pt;
      symbol_table(table_pt).typ := pkg_bdy3;
            symbol_table(table_pt).pk_pt := pl;
            pp := pl;
            display(lvl+1) := table_pt;
            symbol_table(table_pt).sect_end := table_pt;
           end if;
          else
           debug.error(60,symbols'pos(pack1));
          end if;
          nextsym;
         else
          debug.error(61, symbols'pos(id1));
         end if;
         checksym(is1,56);
         pkg_body_decl(lvl+1);
         pkg_search(pp, tp);
        --end when body1
      when others => debug.error(61,symbols'pos(pack1));
   end case;
end;

procedure package_decl(lvl: integer) is
begin
  checksym(pack1, 52);
  pkg_tail(lvl);
end;
```

```
procedure declaration(lvl:integer) is
begin
 case sym is
  when id1 => object_decl(lvl);
  when type1 => type_decl(lvl);
  when proc1 => subprog_decl(lvl);
  when pack1 => package_decl(lvl);
  when others => debug.error(2, symbols'pos(sym));
 end case;
end;

procedure decl_part(lvl: integer) is
begin
   if sym = type1 or sym = id1 or sym = proc1 or sym = pack1 then
     declaration(lvl);
     decl_part(lvl);
   elsif sym /= begin1 then
     debug.error(52, symbols'pos(proc1));
   end if;
end;

procedure program is
begin
  subprog_decl(-1);
end;

begin
  getfile;
  nextsym;
  program;
--  show_table;
  exception
    when text_io.name_error =>
      text_io.put_line("File 'out1' not found");
```

```
    when err_excep =>
      text_io.put_line("Error");
end pass2;
```

4. Pass3: Code Generation.

This pass generates code for a stack machine, namely the a-machine implemented in pass4. The code it produces is not very efficient however this could be overcome with an additional optimization pass. This could be a project in a more advanced course. The code produced for the following factorial program is shown below in mnemonic form. The actual code is produced on a file called CODE and is coded into integers.

```
proc tst20 is
x: int;
 proc fact(n: int; ans: out int) is
 k: int;
 begin
    if n <= 1 then ans := 1;
    else
       fact(n - 1, k);
       ans := n * k;
    end if;
 end;
begin
  fact(5, x);
  write(x:5);
end;
```

Code:

```
  1: mbl3    0
  2: cal3    0   30    4
  3: hlt3
  4: lad3    1    3
  5: lod3
  6: lit3    1
  7: le3
```

```
 8: jz3     13
 9: lai3    1    4
10: lit3    1
11: sto3
12: jmp3    29
13: mbl3    5
14: lad3    1    3
15: lod3
16: lit3    1
17: sub3
18: blmd3   3    1
19: lad3    1    5
20: blmd3   4    1
21: cal3    1    4    6
22: lai3    1    4
23: lad3    1    3
24: lod3
25: lad3    1    5
26: lod3
27: mul3
28: sto3
29: ret3    1
30: mbl3    5
31: lit3    5
32: blmd3   3    1
33: lad3    0    3
34: blmd3   4    1
35: cal3    1    4    6
36: lad3    0    3
37: lod3
38: lit3    5
39: wrti3   1
40: ret3    0
41: endm3
```

We now present pass3:

```
with text_io;
package int_io is new text_io.integer_io(integer);
with text_io; with int_io;
procedure pass3 is
 symtop: constant := 500;
 type symbols is (begin1, end1, dot1, elsif1, becomes1, plusop1, --5
                  minusop1,id1, intlit1, lparenth1, rparenth1, --10
                  semicolon1, comma1, endtext1, out1, if1, --15
                  then1, loop1, exit1, lineno1, error1, lt1, le1, --22
                  gt1, ge1, eq1, ne1 , star1, slash1, and1,  --29
                  or1, not1, colon1, int1, bool1,  array1, --35
                  body1, char1, dotdot1, is1, type1, true1, false1,--42
  else1, of1, mod1, null1, pack1, record1, --48
  when1, while1, proc1, in1, write1, writeln1, read1); --55


 type typekind is (int3, bool3, char3, array3, rec3, proc3, pkg_def3,
   pkg_bdy3, notyp3);
 type idkind is (obj4, typ_id4, param4, proc_id4, pkg_id4, unknown_id4);

 type operation is (add3, and3, bld3, blmd3, blmi3, cal3, div3, --6
    endm3, eq3, --8
    ge3, gt3, hlt3, jmp3, jnz3, jz3, lad3,--15
    lai3, le3, lit3, lod3, lt3, mbl3, mod3,--22
    mul3, ne3, neg3, nop3, not3, or3, rdb3, rdi3, --30
    ret3, sto3, sub3, wrtb3, wrti3); --35

 op_sing, op_doub, op_trip: array(operation) of boolean :=
    (add3 .. wrti3 => false);

 int_pt, bool_pt, char_pt: integer;
```

```
type modekind is (in5, out5);

infil, fout2, listfile, mnfile, codefi, codemn: text_io.file_type;
sym: symbols;
i, linenum, secondsym, errposn : integer;
endfile: boolean;

type item is record
bkwrd, lvl, dsp, pdsp, size, lo: integer;
id_name: integer;
kind: idkind;
            typ: typekind;
trans: modekind;
nrml: boolean;
            typ_point, pk_pt: integer;
            param_end, sect_end: integer;
          end record;
symbol_table: array(0 .. symtop) of item;

double, invisible: array(symbols) of boolean := (begin1 .. read1 => false);
levmax: constant := 25;

display: array(integer range -1 .. levmax) of integer;

subtype str11 is string(1 .. 11);
subtype str20 is string(1 .. 20);
subtype str7 is string(1 .. 7);
subtype str8 is string(1 .. 8);
err_excep, coderr: exception;
table_pt: integer;
pkg_d_stack: array(1 .. 100) of integer;
pkg_d_pt: integer;
type order is record
```

```
    oprtn, opnd1, opnd2, opnd3: integer;
end record;
    codelim: constant:= 500;
    code: array(1 .. codelim) of order;
    codept: integer := 0;
    dispconst: constant := 3;

package debug is
  type oprek is record
    opnm:str7;
    oplen:integer;
end record;
  opmnem: array(operation) of oprek;
  tmnem: array(typekind) of str8;
  procedure emit(n: integer);
  function size(n: integer) return integer;
  function length(st: str11) return integer;
  procedure emit1(k,n: integer);
  procedure emit2(m,n:integer);
  procedure error(m,k: integer);
end debug;

package body debug is
  endmntable: constant := 55;
  outlen, mnemlen: integer;
  mnem: array(integer range 0..endmntable) of str11;

function size(n: integer) return integer is
neg: boolean; nn, len: integer;
begin
  nn := n;
  neg := (nn < 0);
  if neg then nn := -nn; end if;
  if  nn <= 9 then  len := 1;
```

```
  elsif nn <= 99 then len := 2;
  elsif nn <= 999 then len := 3;
  elsif nn <= 9999 then len := 4;
  elsif nn <= 32767 then len := 5;
  end if;
  if neg then return len + 1;
  else return len; end if;
end size;

procedure emit(n: integer) is
sz: integer;
begin
  sz := size(n);
  text_io.put(fout2,' '); int_io.put(fout2,n, sz);
  outlen := outlen + sz + 1;
  if outlen >= 72 then
    text_io.new_line(fout2);
    outlen := 0;
  end if ;
end emit;

function length(st: str11) return integer is
i: integer;
begin
  i := 11;
  while st(i) = ' ' loop
    i := i - 1;
  end loop;
  return i;
end;

procedure emit1(k,n: integer) is
lngth, sz: integer;
begin
```

```
  if k = 1 then
    lngth := length(mnem(n));
    emit(n);
    text_io.put(mnfile, ' '& mnem(n)(1 .. lngth));
    mnemlen := mnemlen + lngth + 1;
    if mnemlen >= 65 then
      text_io.put_line(mnfile,"");
      mnemlen := 0;
    end if;
  else
     emit(n);
     sz := size(n);
     text_io.put(mnfile,' '); int_io.put(mnfile,n,sz);
     mnemlen := mnemlen + sz + 1;
     if mnemlen >= 65 then
       text_io.put_line(mnfile,"");
       mnemlen := 0;
     end if;
  end if;
end emit1;


procedure emit2(m,n:integer) is
begin
  emit1(1,m); emit1(2,n);
end;

procedure error(m,k: integer) is
begin
    emit2(symbols'pos(error1), m);
    errposn := m;
    raise err_excep;
end error;
```

```
begin --package debug initialization section
  mnem(0) := "begin1     ";
  mnem(1) := "end1       ";
  mnem(2) := "dot1       ";
  mnem(3) := "elsif1     ";
  mnem(4) := "becomes1   ";
  mnem(5) := "plusop1    ";
  mnem(6) := "minusop1   ";
  mnem(7) := "id1        ";
  mnem(8) := "intlit1    ";
  mnem(9) := "lparenth1  ";
  mnem(10) := "rparenth1  ";
  mnem(11) := "semicolon1 ";
  mnem(12) := "comma1     ";
  mnem(13) := "endtext1   ";
  mnem(14) := "out1       ";
  mnem(15) := "if1        ";
  mnem(16) := "then1      ";
  mnem(17) := "loop1      ";
  mnem(18) := "exit1      ";
  mnem(19) := "lineno1    ";
  mnem(20) := "error1     ";
  mnem(21) := "lt1        ";
  mnem(22) := "le2        ";
  mnem(23) := "gt1        ";
  mnem(24) := "ge1        ";
  mnem(25) := "eq1        ";
  mnem(26) := "ne1        ";
  mnem(27) := "star1      ";
  mnem(28) := "slash1     ";
  mnem(29) := "and1       ";
  mnem(30) := "or1        ";
  mnem(31) := "not1       ";
```

```
mnem(32) := "colon1    ";
mnem(33) := "int1      ";
mnem(34) := "bool1     ";
mnem(35) := "array1    ";
mnem(36) := "body1     ";
mnem(37) := "char1     ";
mnem(38) := "dotdot1   ";
mnem(39) := "is1       ";
mnem(40) := "type1     ";
mnem(41) := "true1     ";
mnem(42) := "false1    ";
mnem(43) := "else1     ";
mnem(44) := "of1       ";
mnem(45) := "mod1      ";
mnem(46) := "null1     ";
mnem(47) := "pack1     ";
mnem(48) := "record1   ";
mnem(49) := "when1     ";
mnem(50) := "while1    ";
mnem(51) := "proc1     ";
mnem(52) := "in1       ";
mnem(53) := "write1    ";
mnem(54) := "writeln1  ";
mnem(55) := "read1     ";

opmnem  := (("add3   ", 4),
   ("and3   ", 4),
   ("bld3   ", 4),
   ("blmd3  ", 5),
   ("blmi3  ", 5),
   ("cal3   ", 4),
   ("div3   ", 4),
   ("endm3  ", 5),
   ("eq3    ", 3),
```

```
        ("ge3    ", 3),
        ("gt3    ", 3),
        ("hlt3   ", 4),
        ("jmp3   ", 4),
        ("jnz3   ", 4),
        ("jz3    ", 3),
        ("lad3   ", 4),
        ("lai3   ", 4),
        ("le3    ", 3),
        ("lit3   ", 4),
        ("lod3   ", 4),
        ("lt3    ", 3),
        ("mbl3   ", 4),
        ("mod3   ", 5),
        ("mul3   ", 4),
        ("ne3    ", 3),
        ("neg3   ", 4),
        ("nop3   ", 4),
        ("not3   ", 4),
        ("or3    ", 3),
        ("rdb3   ", 4),
        ("rdi3   ", 4),
        ("ret3   ", 4),
        ("sto3   ", 4),
        ("sub3   ", 4),
        ("wrtb3  ", 5),
        ("wrti3  ", 5));
    tmnem := ("int3    ", "bool3   ", "char3   ",
      "array3  ", "rec3    ", "proc3   ", "pkg_def3",
      "pkg_bdy3", "notyp3  ");

    outlen := 0; mnemlen := 0;
end debug;
```

```
package initialize is
  procedure init;
end initialize;

package body initialize is
  procedure init is
  begin
    double(id1) := true;
    double(intlit1) := true;
    double(lineno1) := true;
    double(error1) := true;
    invisible(lineno1) := true;
    invisible(error1) := true;
    pkg_d_pt := 0;
    op_sing(lit3) := true;
    op_sing(mbl3) := true;
    op_sing(jmp3) := true;
    op_sing(jnz3) := true;
    op_sing(jz3) := true;
    op_sing(wrti3) := true;
    op_sing(wrtb3) := true;
    op_trip(cal3) := true;
    op_doub(lad3) := true;
    op_doub(lai3) := true;
    op_sing(bld3) := true;
    op_doub(blmd3) := true;
    op_doub(blmi3) := true;
    op_sing(ret3) := true;
  end init;
begin
  init;
end initialize;

procedure coder0(oprt: operation) is
```

```
begin
    codept := codept + 1;
    if codept > codelim then
      raise coderr;
    end if;
    code(codept).oprtn := operation'pos(oprt);
end;

procedure coder1(oprt: operation; opd: integer) is
begin
   codept := codept + 1;
   if codept > codelim then
     raise coderr;
   end if;
   code(codept).oprtn := operation'pos(oprt);
   code(codept).opnd1 := opd;
end;

procedure coder2(oprt: operation; opd1, opd2: integer) is
begin
   codept := codept + 1;
   if codept > codelim then
     raise coderr;
   end if;
   code(codept).oprtn := operation'pos(oprt);
   code(codept).opnd1 := opd1;
   code(codept).opnd2 := opd2;
end;

procedure coder3(oprt: operation; opd1, opd2, opd3: integer) is
begin
   codept := codept + 1;
   if codept > codelim then
     raise coderr;
```

```
    end if;
    code(codept).oprtn := operation'pos(oprt);
    code(codept).opnd1 := opd1;
    code(codept).opnd2 := opd2;
    code(codept).opnd3 := opd3;
end;

procedure show_ops is
op: operation;
begin
  for i in 1 .. codept loop
    int_io.put(i,4);text_io.put(":  ");
    op := operation'val(code(i).oprtn);
    text_io.put(debug.opmnem(op).opnm);
    if op_sing(op) then
      int_io.put(code(i).opnd1,6);
    elsif op_doub(op) then
      int_io.put(code(i).opnd1,6);
      int_io.put(code(i).opnd2,6);
    elsif op_trip(op) then
      int_io.put(code(i).opnd1,6);
      int_io.put(code(i).opnd2,6);
      int_io.put(code(i).opnd3,6);
    end if;
    text_io.new_line;
  end loop;
  text_io.put("codept = ");
  int_io.put(codept);
  text_io.new_line;
end;

procedure  getfile is
 use text_io;
begin
```

```
  open(infil,in_file,"out1");
  create(listfile,out_file,"lister");
  create(fout2,out_file,"out2");
  -- The file mnem2 is used for debugging purposes.
  -- It can be eliminated later.
  create(mnfile, out_file, "mnem2");
  text_io.put_line(mnfile,"pass3.ada mnem2");
  text_io.new_line(mnfile);
  create(codefi, text_io.out_file, "code");
end;


procedure nextsym is
firstsym: integer;
use int_io;
begin
  get(infil,firstsym);
  sym := symbols'val(firstsym);
  while invisible(sym) loop
    debug.emit1(1,firstsym);
    get(infil, secondsym);
    debug.emit1(2, secondsym);
    if sym = lineno1 then
      linenum := secondsym;
    elsif sym = error1 then
      debug.error(1, secondsym);
    end if;
    get(infil, firstsym);
    sym := symbols'val(firstsym);
  end loop;
  debug.emit1(1,firstsym);
  if double(sym) then
    get(infil, secondsym);
    debug.emit1(2, secondsym);
```

```
  end if;
end nextsym;

package b_entry is
procedure basic_entry(id,tpt: integer;
                knd:idkind; tp:typekind);
end b_entry;

package body b_entry is
procedure basic_entry(id,tpt: integer;
                knd:idkind; tp:typekind) is
begin
  table_pt := table_pt + 1;
  symbol_table(table_pt).bkwrd := table_pt - 1;
  symbol_table(table_pt).id_name := id;
  symbol_table(table_pt).lvl := -1;
  symbol_table(table_pt).kind := knd;
  symbol_table(table_pt).typ := tp;
  symbol_table(table_pt).typ_point := tpt;
end;

begin -- b_entry
  table_pt := 0; display(-1) := 1;
  basic_entry(-1, -1, proc_id4, proc3);
  basic_entry(symbols'pos(int1), -1, typ_id4, int3);
  int_pt := table_pt;
  symbol_table(table_pt).size := 1;
  basic_entry(symbols'pos(bool1), -1, typ_id4, bool3);
  bool_pt := table_pt;
  symbol_table(table_pt).size := 1;
  basic_entry(symbols'pos(char1), -1, typ_id4, char3);
  char_pt := table_pt;
  symbol_table(table_pt).size := 1;
  symbol_table(1).sect_end := table_pt;
```

```
end b_entry;

procedure checksym(symb: symbols; n: integer) is
begin
  if sym = symb then nextsym;
  else debug.error(n,symbols'pos(sym)); end if;
end;

procedure checksemi(n: integer) is
begin
 checksym(semicolon1,n);
end;

procedure show_table is
begin
 text_io.put(" inx    name    type   typpt    bwrd    sctnd");
 text_io.put_line(" dsp    size     lvl     lo");
 for i in 1 .. table_pt loop
  int_io.put(i,3);
  int_io.put(symbol_table(i).id_name,9);
  text_io.put("  ");
  text_io.put(debug.tmnem(symbol_table(i).typ));
  int_io.put(symbol_table(i).typ_point,7);
  int_io.put(symbol_table(i).bkwrd,7);
  int_io.put(symbol_table(i).sect_end,7);
  int_io.put(symbol_table(i).dsp,8);
  int_io.put(symbol_table(i).size,8);
  int_io.put(symbol_table(i).lvl,9);
  int_io.put(symbol_table(i).lo,7);
  text_io.new_line;
 end loop;
end;

procedure show_table2 is
```

```
begin
  text_io.put(" inx    name    type  pdsp    bwrd    sctnd");
  text_io.put_line("  dsp    size     lvl     lo");
  for i in 1 .. table_pt loop
    int_io.put(i,3);
    int_io.put(symbol_table(i).id_name,9);
    text_io.put("   ");
    text_io.put(debug.tmnem(symbol_table(i).typ));
    int_io.put(symbol_table(i).pdsp,7);
    int_io.put(symbol_table(i).bkwrd,7);
    int_io.put(symbol_table(i).sect_end,7);
    int_io.put(symbol_table(i).dsp,8);
    int_io.put(symbol_table(i).size,8);
    int_io.put(symbol_table(i).lvl,9);
    int_io.put(symbol_table(i).lo,7);
    text_io.new_line;
  end loop;
end;

procedure enter_var(iden, lv, plvl: integer; knd: idkind) is
j, k: integer;
begin
  table_pt := table_pt + 1;
  symbol_table(table_pt).bkwrd :=
                symbol_table(display(lv)).sect_end;
  symbol_table(table_pt).id_name := iden;
  symbol_table(table_pt).lvl := plvl;
  symbol_table(table_pt).kind := knd;
  symbol_table(table_pt).typ := notyp3;
  symbol_table(table_pt).nrml := true;
  symbol_table(display(lv)).sect_end := table_pt;
  symbol_table(table_pt).pk_pt := -1;
end;
```

```
procedure one_sect_check(ss,tp: integer; pl: out integer) is
temp, j: integer;
-- checks a section for an id
begin
  temp := symbol_table(tp).bkwrd;
  symbol_table(tp).bkwrd := 0;
  symbol_table(0).id_name := ss;
  j := symbol_table(tp).sect_end;
  while symbol_table(j).id_name /= ss loop
    j := symbol_table(j).bkwrd;
  end loop;
  symbol_table(tp).bkwrd := temp;
  pl := j;
end;


procedure check_new_id(n, ss, lvl: integer; pl : out integer) is
temp, j, k, pp, tp: integer;
begin
  tp := display(lvl);
  symbol_table(0).id_name := ss;
  one_sect_check(ss,tp,j);
-- If the section is a pack body then special conditions prevail.
-- For object variables and type variables check_new_id should consider
-- the pack def as part of the search domain. In the case of procedures
-- however check_new_id should only search within the pack body. A second
-- check later will verify that the pack def procedures have a body.
 if n = 1 then
  if (j = 0) and (symbol_table(tp).typ = pkg_bdy3) then
     pp := symbol_table(display(lvl)).pk_pt;
     one_sect_check(ss, pp, k);
  end if;
 end if;
  pl := j;
```

```
end;

function id_index(id, lvl: integer) return integer is
lv, lnk, temp, tp, pp: integer;
begin
  lv := lvl; symbol_table(0).id_name := id;
  loop
    tp := display(lv);
    one_sect_check(id,tp,lnk);
    if (lnk = 0) and (symbol_table(tp).typ = pkg_bdy3) then
      pp := symbol_table(tp).pk_pt;
      one_sect_check(id,pp,lnk);
    end if;
    lv := lv - 1;
    exit when (lv < - 1) or (lnk /= 0);
  end loop;
  return lnk;
end;

procedure id_list_tail(lvl, plvl: integer) is
place: integer;
begin
  if sym = comma1 then
    nextsym;
    if sym = id1 then
      check_new_id(1, secondsym,lvl, place);
      if place = 0 then enter_var(secondsym,lvl,plvl,obj4);
      else debug.error(7,symbols'pos(id1)); end if;
      nextsym;
    else debug.error(8,symbols'pos(id1)); end if;
    id_list_tail(lvl, plvl);
  else
    -- test follow symbols for id_list_tail
    if sym /= colon1 then debug.error(6,symbols'pos(colon1));
```

```
    end if;
  end if;
end;

procedure id_list(lvl, plvl: integer) is
place : integer;
begin
 if sym = id1 then
   check_new_id(1, secondsym, lvl, place);
   if place = 0 then enter_var(secondsym,lvl,plvl, obj4);
   else debug.error(5,symbols'pos(id1)); end if;
   nextsym;
 else debug.error(6,symbols'pos(id1)); end if;
 id_list_tail(lvl, plvl);
end;

procedure type_ind(lvl: integer; tp: out integer) is
j: integer;
begin
  case sym is
    when id1 =>
j:= id_index(secondsym, lvl);
if j = 0 then debug.error(9,symbols'pos(id1));
elsif symbol_table(j).kind /= typ_id4 then
  debug.error(10,symbols'pos(id1));
else tp := j;
end if;
nextsym;
    when int1 =>
   tp := 2; nextsym;
    when bool1 =>
   tp := 3; nextsym;
    when char1 =>
   tp := 4; nextsym;
```

```
      when others =>
debug.error(11,symbols'pos(id1));
   end case;
end;

procedure fill_type(st, fin, tp: integer; displ: in out integer) is
tt: typekind; disp, sz: integer;
begin
  disp := displ;
  tt := symbol_table(tp).typ;
  sz := symbol_table(tp).size;
  for i in st .. fin loop
    symbol_table(i).typ_point := tp;
    symbol_table(i).typ := tt;
    symbol_table(i).dsp := disp;
    symbol_table(i).size := sz;
    if symbol_table(i).typ = array3 then
      symbol_table(i).lo := symbol_table(tp).lo;
    end if;
    disp := disp + sz;
  end loop;
  displ := disp;
end;

procedure object_decl(lvl, plvl: integer; displ: in out integer) is
strt_pt, end_pt, tp_pt: integer;
begin
   strt_pt := table_pt + 1;
   id_list(lvl, plvl);
   end_pt := table_pt;
   checksym(colon1, 3);
   type_ind(lvl,tp_pt);
   fill_type(strt_pt, end_pt, tp_pt, displ);
   checksemi(4);
```

```
end;

procedu e stype_def(lvl: integer) is
se: integer;
begin
  se := symbol_table(display(lvl)).sect_end;
  case sym is
    when int1 =>  symbol_table(se).typ_point := int_pt;
                  symbol_table(se).typ := int3;
    when bool1 =>  symbol_table(se).typ_point := bool_pt;
                  symbol_table(se).typ := bool3;
    when others => debug.error(17, symbols'pos(type1));
  end case;
  nextsym;
end;

procedure index_bd(val: out integer) is
begin
  case sym is
    when intlit1 =>
val := secondsym;
nextsym;
    when plusop1 =>
nextsym;
if sym = intlit1 then
 val := secondsym;
 nextsym;
else
 debug.error(230, symbols'pos(intlit1));
end if;
    when minusop1 =>
nextsym;
if sym = intlit1 then
 val := - secondsym;
```

```
  nextsym;
else
  debug.error(2301, symbols'pos(intlit1));
end if;
    when others =>
debug.error(231,symbols'pos(intlit1));
  end case;
end;

procedure index_range(low, high: out integer) is
begin
   index_bd(low);
   checksym(dotdot1, 23);
   index_bd(high);
end;

procedure array_def(lvl: integer) is
tp, j, low, high: integer;
begin
  checksym(array1, 18);
  checksym(lparenth1, 19);
  index_range(low, high);
  symbol_table(table_pt).lo := low;
  checksym(rparenth1,  21);
  checksym(of1, 20);
  type_ind(lvl, tp);
  j := symbol_table(display(lvl)). sect_end;
  symbol_table(j).typ_point := tp;
  symbol_table(j).typ := array3;
  symbol_table(j).size := (high - low + 1) * symbol_table(tp).size;
end;

procedure comp_def_tail(lvl, plvl: integer; displ: in out integer) is
begin
```

```
  if sym = id1 then
    object_decl(lvl, plvl, displ);
    comp_def_tail(lvl, plvl, displ);
  elsif sym /= end1 then  -- takes care of follow symbols
    debug.error(29,symbols'pos(id1));
  end if;
end;

procedure comp_def(lvl, plvl: integer) is
disp: integer;
begin
  disp := 0;
  object_decl(lvl, plvl, disp);
  comp_def_tail(lvl, plvl, disp);
  symbol_table(display(lvl)).size := disp;
end;


procedure record_def(lvl, plvl: integer) is
tp, disp: integer;
begin
  checksym(record1, 25);
  symbol_table(table_pt).typ := rec3;
  tp := table_pt;
  symbol_table(tp).sect_end := tp;
  symbol_table(tp).typ_point := tp;
  display(lvl+1) := tp;
  comp_def(lvl+1, -1);
  checksym(end1, 27);
  checksym(record1, 28);
  symbol_table(tp).sect_end := table_pt;
end;

procedure type_def(lvl, plvl: integer) is
```

```
begin
  case sym is
   when int1 | bool1 | char1 => stype_def(lvl);
   when array1 => array_def(lvl);
   when record1 => record_def(lvl, plvl);
   when others => debug.error(16,symbols'pos(type1));
  end case;
end;

procedure type_decl(lvl, plvl: integer) is
pl: integer;
begin
   checksym(type1,12);
   if sym = id1 then
     check_new_id(1, secondsym,lvl, pl);
     if pl /= 0 then
       debug.error(13,symbols'pos(id1));
     else
       enter_var(secondsym, lvl, plvl, typ_id4);
     end if;
     symbol_table(table_pt).dsp := 0;
     nextsym;
   else
     debug.error(14,symbols'pos(id1));
   end if;
   checksym(is1,14);
   type_def(lvl, plvl);
   checksemi(15);
end;

procedure mode(md: out modekind) is
begin
   case sym is
     when in1 => md := in5; nextsym;
```

```
    when out1 => md := out5; nextsym;
    -- deal with follow symbols
    when id1 | int1 | bool1  => md := in5;
    when others => debug.error(46,symbols'pos(type1));
  end case;
end;

procedure p_tail(lvl, plvl: integer) is
place: integer;
begin
 if sym = comma1 then
    checksym(comma1,40);
    if sym = id1 then
check_new_id(2, secondsym, lvl, place);
if place = 0 then enter_var(secondsym,lvl, plvl, param4);
else debug.error(41,symbols'pos(id1)); end if;
nextsym;
p_tail(lvl, plvl);
    else debug.error(42,symbols'pos(id1)); end if;
 elsif sym /= colon1 then
    debug.error(43, symbols'pos(id1));
 end if;
end;

procedure sparam_decl(lvl, plvl: integer; disp: in out integer) is
strt_pt, end_pt, tp, i: integer; tt:typekind;
md: modekind; place, displ, sz, ss: integer;
begin
 displ := disp;
 if sym = id1 then
   check_new_id(2, secondsym, lvl, place);
   if place = 0 then
     strt_pt := table_pt + 1;
     enter_var(secondsym,lvl,plvl, param4);
```

```
    else debug.error(37,symbols'pos(id1)); end if;
    nextsym;
  else debug.error(38,symbols'pos(id1)); end if;
  p_tail(lvl, plvl);
  end_pt := table_pt;
  checksym(colon1,39);
  mode(md);
  type_ind(lvl, tp);
  tt := symbol_table(tp).typ;
  sz := symbol_table(tp).size;
  for i in strt_pt .. end_pt loop
    symbol_table(i).typ_point := tp;
    symbol_table(i).typ := tt;
    symbol_table(i).trans := md;
    if md = out5 then
      symbol_table(i).nrml := false;
      ss := 1;
    else
      ss := sz;
    end if;
    symbol_table(i).dsp := displ;
    symbol_table(i).size := ss;
    displ := displ + ss;
  end loop;
  disp := displ;
end;

procedure p_dec_tail(lvl, plvl: integer; displ: in out integer) is
begin
  if sym = semicolon1 then
      checksemi(44);
      sparam_decl(lvl, plvl, displ);
      p_dec_tail(lvl, plvl, displ);
  elsif sym /= rparenth1 then
```

```
    debug.error(45, symbols'pos(rparenth1));
  end if;
end;

procedure param_decl(lvl, plvl: integer; displ: in out integer) is
begin
  sparam_decl(lvl, plvl, displ);
  p_dec_tail(lvl,plvl, displ);
end;

procedure formal_part(lvl, plvl: integer; displ: in out integer) is
begin
 if sym = lparenth1 then
    checksym(lparenth1,35);
    param_decl(lvl, plvl, displ);
    checksym(rparenth1,36);
 end if;
end;

procedure subprog_header(lvl, plvl: integer; disp, tpt: out integer) is
pl, tp, displ: integer;
begin
  displ := dispconst;
  checksym(proc1,32);
  if sym = id1 then
    tp := table_pt + 1;
    enter_var(secondsym, lvl, plvl, proc_id4);
    symbol_table(tp).sect_end := tp;
    symbol_table(tp).param_end := tp;
    symbol_table(tp).typ := proc3;
    symbol_table(tp).size := -1;
    symbol_table(tp).lo := -1;
    tpt := tp;
    nextsym;
```

```
    end if;
    display(lvl+1) := tp;
    formal_part(lvl+1, plvl+1, displ);
    symbol_table(tp).pdsp := displ;
    disp := displ;
    symbol_table(tp).param_end :=
      symbol_table(tp).sect_end;
end;

procedure expression(lvl, plvl: integer; typ_pt: out integer);

procedure vbl(lvl, plvl: integer; typ_pt: out integer);

procedure factor(lvl, plvl: integer; typ_pt: out integer) is
tpt: integer;
begin
  case sym is
    when intlit1 =>
typ_pt := int_pt;
coder1(lit3, secondsym);
nextsym;
    when lparenth1 =>
nextsym;
expression(lvl, plvl, typ_pt);
checksym(rparenth1, 160);
    when id1 =>
vbl(lvl, plvl, tpt);
if (tpt = int_pt) or (tpt = bool_pt) then
  coder0(lod3);
end if;
typ_pt := tpt;
    when not1 =>
nextsym;
factor(lvl, plvl, tpt);
```

```
coder0(not3);
typ_pt := bool_pt;
    when true1 | false1 =>
if sym = true1 then
 coder1(lit3, 1);
else
 coder1(lit3, 0);
end if;
nextsym;
typ_pt := bool_pt;
    when others => debug.error(162,symbols'pos(id1));
  end case;
end;

procedure term_tail(lvl, plvl, typ_pt: integer) is
tpt1, tpt2: integer; arith_op: boolean; sym2: symbols;
begin
   case sym is
    when and1 | star1 | mod1 | slash1 =>
 sym2 := sym;
 nextsym;
 factor(lvl, plvl, tpt2);
 term_tail(lvl, plvl, typ_pt);
 case sym2 is
  when and1 => coder0(and3);
  when star1 => coder0(mul3);
  when mod1 =>  coder0(mod3);
  when slash1 => coder0(div3);
  when others => null;
 end case;
    when colon1 | semicolon1 | rparenth1 | comma1 | eq1 | ne1 |
 lt1 | le1 | gt1 | ge1 | plusop1 | minusop1 | or1 |
 then1 | loop1 => null;
    when others => debug.error(165, symbols'pos(and1));
```

```
    end case;
end;

procedure term(lvl, plvl: integer; typ_pt: out integer) is
tpt: integer;
begin
  factor(lvl, plvl, tpt);
  term_tail(lvl, plvl, tpt);
  typ_pt := tpt;
end;

procedure sexp_tail(lvl, plvl, typ_pt: integer) is
tpt1, tpt2: integer; sym2: symbols;
begin
  case sym is
    when plusop1 | minusop1 | or1 =>
sym2 := sym;
nextsym;
term(lvl, plvl, tpt2);
sexp_tail(lvl, plvl, tpt2);
case sym2 is
 when plusop1 => coder0(add3);
 when minusop1 => coder0(sub3);
 when or1 => coder0(or3);
 when others => null;
end case;
    when colon1 | semicolon1 | rparenth1 | comma1 | eq1 | ne1 |
  lt1 | le1 | gt1 | ge1 | then1 | loop1 => null;
    when others =>
debug.error(168, symbols'pos(or1));
  end case;
end;

procedure simp_expr(lvl, plvl: integer; typ_pt: out integer) is
```

```
tpt1, tpt2: integer; aflag: boolean; sym2: symbols;
begin
  case sym is
   when plusop1 | minusop1 =>
     aflag := true;
     sym2 := sym;
     nextsym;
   when others =>
     aflag := false;
  end case;
  term(lvl, plvl, tpt1);
  if aflag and (tpt1 /= int_pt) then
   debug.error(169, symbols'pos(plusop1));
  end if;
  if aflag then
   if sym2 = minusop1 then
     coder0(neg3);
   end if;
  end if;
  sexp_tail(lvl, plvl, tpt1);
  typ_pt := tpt1;
end;

procedure exp_tail(lvl, plvl: integer; typ_pt: in out integer) is
tpt: integer; bflag: boolean; sym2: symbols;
begin
    case sym is
when eq1 | ne1 | lt1 | le1 | gt1 | ge1  =>
  sym2 := sym;
  nextsym;
  simp_expr(lvl, plvl, tpt);
  if typ_pt /= tpt then
   debug.error(171,symbols'pos(eq1));
  elsif (tpt /= int_pt) and (tpt /= bool_pt) then
```

```
    debug.error(300,symbols'pos(eq1));
  else
    typ_pt := bool_pt;
  end if;
  case sym2 is
    when eq1 => coder0(eq3);
    when ne1 => coder0(ne3);
    when lt1 => coder0(lt3);
    when le1 => coder0(le3);
    when gt1 => coder0(gt3);
    when ge1 => coder0(ge3);
    when others => null;
  end case;
when colon1 | semicolon1 | rparenth1 | comma1 | then1 | loop1 =>
  null;
when others =>
  debug.error(172,symbols'pos(eq1));
    end case;
end;

procedure expression(lvl, plvl: integer; typ_pt: out integer) is
tpt1, tpt2: integer;
begin
  simp_expr(lvl, plvl, tpt1);
  tpt2 := tpt1;
  exp_tail(lvl, plvl, tpt2);
  typ_pt := tpt2;
end;

procedure vbl_tail(lvl, plvl:integer; typ_pt: in out integer) is
tpt: integer;
begin
   case sym is
    when lparenth1 =>
```

```
  checksym(lparenth1, 120);
  expression(lvl, plvl, tpt);
  if symbol_table(typ_pt).lo /= 0 then
    coder1(lit3, symbol_table(typ_pt).lo);
    coder0(sub3);
  end if;
  checksym(rparenth1, 140);
  typ_pt := symbol_table(typ_pt).typ_point;
  if symbol_table(typ_pt).size > 1 then
    coder1(lit3, symbol_table(typ_pt).size);
    coder0(mul3);
  end if;
  coder0(add3);
  vbl_tail(lvl, plvl, typ_pt);
     when  colon1 | semicolon1 | rparenth1 | comma1 | eq1 | ne1 |
   lt1 | le1 | gt1 | ge1 | plusop1 | minusop1 | and1 |
   star1 | mod1 | slash1 | or1 | then1 | loop1 | dot1 | becomes1
   => null;
     when others =>
   debug.error(122, symbols'pos(id1));
    end case;
end;

procedure vbl1(lvl, plvl, pl: integer; typ_pt: out integer);

procedure v1_tail(lvl, plvl, pl, pl0: integer; typ_pt :out integer) is
pl1, pl2: integer;
begin
  case sym is
    when dot1 =>
      checksym(dot1,117);
      if sym /= id1 then
  debug.error(119, symbols'pos(id1));
      else
```

```
  pl1 := symbol_table(pl).typ_point;
  one_sect_check(secondsym,pl1,pl2);
  vbl1(lvl, plvl, pl2, typ_pt);
      end if;
    when colon1 | semicolon1 | rparenth1 | comma1 | eq1 | ne1 | lt1 | le1 |
 gt1 | ge1 | plusop1 | minusop1 | and1 | star1 | mod1 |
 slash1 | or1 | then1 | loop1 | becomes1  => typ_pt := pl0;
    when others =>  debug.error(205, symbols'pos(id1));
  end case;
end;

procedure vbl1(lvl, plvl, pl: integer; typ_pt: out integer) is
dummy, tpt1, tpt2: integer;
begin
if symbol_table(pl).lvl /= -1 then
  if symbol_table(pl).nrml then
    coder2(lad3,symbol_table(pl).lvl, symbol_table(pl).dsp);
  else
    coder2(lai3,symbol_table(pl).lvl, symbol_table(pl).dsp);
  end if;
else
  coder1(lit3, symbol_table(pl).dsp);
  coder0(add3);
end if;
dummy := 0;
nextsym;
case symbol_table(pl).typ is
   when rec3 =>
      tpt1 := symbol_table(pl).typ_point;
      v1_tail(lvl, plvl, pl, tpt1, tpt2);
      typ_pt := tpt2;
   when int3 =>
      vbl_tail(lvl, plvl, dummy);
      typ_pt := int_pt;
```

```
   when bool3 =>
      vbl_tail(lvl, plvl, dummy);
      typ_pt := bool_pt;
   when array3 =>
      tpt1 := symbol_table(pl).typ_point;
      vbl_tail(lvl, plvl, tpt1);
      vl_tail(lvl, plvl, tpt1, tpt1, tpt2);
      typ_pt := tpt2;
   when others => debug.error(141,symbols'pos(id1));
 end case;
end;

procedure vbl2(lvl, plvl, pl: integer; typ_pt: out integer) is
pl1: integer;
begin
    nextsym;
    checksym(dot1, 127);
    one_sect_check(secondsym,pl,pl1);
    vbl1(lvl, plvl, pl1, typ_pt);
end;

procedure vbl(lvl, plvl: integer; typ_pt: out integer) is
pl, pl1: integer;
begin
    pl := id_index(secondsym, lvl);
    case symbol_table(pl).kind is
      when obj4 | param4 => vbl1(lvl, plvl, pl, typ_pt);
      when pkg_id4 =>
  pl1 := symbol_table(pl).pk_pt;
  vbl2(lvl, plvl, pl1, typ_pt);
      when others => debug.error(131, symbols'pos(id1));
    end case;
end;
```

```
procedure exit_stat(lvl, plvl: integer; cp: in out integer) is
tpt: integer;
begin
  checksym(exit1,103);
  checksym(when1,104);
  expression(lvl, plvl, tpt);
  if symbol_table(tpt).typ /= bool3 then
    debug.error(105,symbols'pos(bool1));
  end if;
  coder1(jnz3, cp);
  cp := codept;
end;

procedure p_sub_tail(lvl, plvl, pl, pl0: integer);

procedure exp_p_sub(lvl, plvl, pl, pl1: integer) is
tpt1, pl2, sz: integer;
begin
  if symbol_table(pl1).trans = in5 then
    expression(lvl, plvl, tpt1);
    sz := symbol_table(tpt1).size;
    if sz > 1 then
      coder2(blmi3,symbol_table(pl1).dsp, sz);
    else
      coder2(blmd3, symbol_table(pl1).dsp, 1);
    end if;
    p_sub_tail(lvl, plvl, pl, pl1);
  elsif symbol_table(pl1).trans = out5 then  --redundant check
    vbl(lvl, plvl, tpt1);
    coder2(blmd3,symbol_table(pl1).dsp, 1);
    p_sub_tail(lvl, plvl, pl, pl1);
  end if;
end;
```

```
procedure p_sub_tail(lvl, plvl, pl, pl0: integer) is
tpt1, pl1, pl2: integer;
begin
  case sym is
    when comma1 =>
nextsym;
pl1 := pl0 + 1;
exp_p_sub(lvl, plvl, pl, pl1);
    when rparenth1 =>
      if pl0 /= symbol_table(pl).param_end then
  debug.error(198, symbols'pos(id1));
      end if;
    when others => debug.error(197, symbols'pos(id1));
  end case;
end;

procedure p_call_tail(lvl, plvl, pl: integer) is
tpt1, pl1, pl2: integer;
begin
  case sym is
    when lparenth1 =>
nextsym;
pl1 := pl + 1;
exp_p_sub(lvl,plvl,pl,pl1);
checksym(rparenth1, 182);
    when semicolon1 =>
      if pl /= symbol_table(pl).param_end then
  debug.error(188, symbols'pos(id1));
      end if;
    when others => debug.error(187, symbols'pos(id1));
  end case;
end;

procedure proc_call_stat(lvl, plvl, pl: integer) is
```

```
addr, pl1, pl2: integer;
begin
--pl points to symbol table entry for id
--which is a proc id
   nextsym;
-- prepare for call-fix-up
-- size is used to maintain fix-up information
-- and start of code address
   coder1(mbl3, symbol_table(pl).pdsp);
   p_call_tail(lvl, plvl, pl);
   addr := symbol_table(pl).size;
   if symbol_table(pl).lo = -1 then
     if symbol_table(pl).size = -1 then
symbol_table(pl).size := codept + 1;
     else
pl1 := symbol_table(pl).size;
while pl1 /= -1 loop
  pl2 := pl1;
  pl1 := code(pl1).opnd2;
end loop;
code(pl2).opnd2 := codept + 1;
     end if;
     coder3(cal3, symbol_table(pl).lvl+1, -1,-1);
    else
      coder3(cal3, symbol_table(pl).lvl+1, symbol_table(pl).size,
 symbol_table(pl).dsp);
    end if;
end;

procedure assign_stat(lvl, plvl, pl: integer) is
typ1, typ2: integer;
begin
   vbl1(lvl, plvl, pl, typ1);
   checksym(becomes1, 97);
```

```
    expression(lvl, plvl, typ2);
    if symbol_table(typ2).size = 1 then
      coder0(sto3);
    else
      coder1(bld3, symbol_table(typ2).size);
    end if;
end;

procedure pk_stat_tail(lvl, plvl, pl: integer) is
pl1: integer;
begin
    one_sect_check(secondsym,pl,pl1);
     case symbol_table(pl1).kind is
when proc_id4 => proc_call_stat(lvl, plvl, pl1);
when obj4 | param4 => assign_stat(lvl, plvl, pl1);
       when others => debug.error(94, symbols'pos(pack1));
     end case;
end;


procedure pk_stat(lvl, plvl, pl: integer) is
begin
  nextsym;
  checksym(dot1,93);
  pk_stat_tail(lvl, plvl, pl);
end;

procedure read_stat(lvl, plvl: integer) is
tpt: integer;
begin
  nextsym;
  checksym(lparenth1, 306);
  vbl(lvl, plvl, tpt);
  if tpt = int_pt then
```

```
      coder0(rdi3);
    elsif tpt = bool_pt then
      coder0(rdb3);
    end if;
    checksym(rparenth1, 320);
end;


procedure w_tail(lvl, plvl, tp: integer) is
tpt, f: integer;
begin
    case sym is
      when colon1 => nextsym;
expression(lvl, plvl, tpt);
if tpt /= int_pt then
  debug.error(305, symbols'pos(id1));
end if;
f := 1;
      when rparenth1 => f := 0;
      when others => debug.error(304, symbols'pos(id1));
    end case;
       if tp = int_pt then
 coder1(wrti3, f);
       elsif tp = bool_pt then
 coder1(wrtb3, f);
       else debug.error(3021,200);
       end if;
end;

procedure write_body(lvl, plvl: integer) is
tpt: integer;
begin
    checksym(lparenth1, 301);
    expression(lvl, plvl, tpt);
```

```
  if (tpt /= int_pt) and (tpt /= bool_pt) then
    debug.error(302, symbols'pos(id1));
  end if;
  w_tail(lvl, plvl, tpt);
  checksym(rparenth1,303);
end;

procedure write_stat(lvl, plvl: integer) is
begin
  case sym is
    when write1 =>
      nextsym;
      write_body(lvl, plvl);
    when writeln1 =>
      nextsym;
    when others => null;
  end case;
end;

procedure simp_stat(lvl, plvl: integer) is
pl, pl1, pl2: integer;
begin
  case sym is
    when null1 =>
 checksym(null1,901);
 coder0(nop3);
    when id1 =>
       pl := id_index(secondsym,lvl);
       case symbol_table(pl).kind is
  when proc_id4 => proc_call_stat(lvl, plvl, pl);
  when obj4 | param4 => assign_stat(lvl, plvl, pl);
  when pkg_id4 =>
pl := symbol_table(pl).pk_pt; -- make sure it points to
    -- pack def
```

```
pk_stat(lvl, plvl, pl);
  when others => debug.error(91,symbols'pos(id1));
      end case;
    when semicolon1 => null;
    when write1 | writeln1 => write_stat(lvl, plvl);
    when read1 => read_stat(lvl, plvl);
    when others => debug.error(92,symbols'pos(id1));
   end case;
end;

procedure stat_seq(lvl, plvl: integer);

procedure sing_elsif_part(lvl, plvl: integer; cp: in out integer) is
tpt, cp1: integer;
begin
     checksym(elsif1, 331);
     expression(lvl, plvl, tpt);
     coder1(jz3, -1);
     cp1 := codept;
     if tpt /= bool_pt then
debug.error(151,symbols'pos(elsif1));
     end if;
     checksym(then1,152);
     stat_seq(lvl, plvl);
     coder1(jmp3,cp);
     cp := codept;
     code(cp1).opnd1 := codept+1;
end;

procedure elsif_part(lvl, plvl: integer; cp: in out integer) is
begin
  case sym is
    when elsif1 =>
 sing_elsif_part(lvl, plvl, cp);
```

```
 elsif_part(lvl, plvl, cp);
    when end1 | else1 => null;
    when others =>
debug.error(330, symbols'pos(elsif1));
  end case;
end;

procedure else_part(lvl, plvl: integer) is
begin
  case sym is
    when else1 =>
nextsym;
stat_seq(lvl, plvl);
    when end1 => null;
    when others => debug.error(154, symbols'pos(else1));
  end case;
end;

procedure if_stat(lvl, plvl: integer) is
tpt, cp1, cp2, cp3, cp4 : integer;
begin
  checksym(if1,145);
  expression(lvl, plvl, tpt);
  coder1(jz3,-1);
  cp1 := codept;
  if tpt /= bool_pt then
   debug.error(146, symbols'pos(if1));
  end if;
  checksym(then1, 1451);
  stat_seq(lvl, plvl);
  coder1(jmp3, -1);
  code(cp1).opnd1 := codept+1;
  cp2 :=codept;
  elsif_part(lvl, plvl, cp2);
```

```
  else_part(lvl, plvl);
  cp3 := cp2;
  while cp3 /= -1 loop
    cp4 := cp3;
    cp3 := code(cp3).opnd1;
    code(cp4).opnd1 := codept+1;
  end loop;
  checksym(end1,147);
  checksym(if1,148);
end;

procedure lstat_seq(lvl, plvl: integer; cp: in out integer);

procedure loop_stat(lvl, plvl, cp, cpw: integer) is
cp0, cp1, cp2, cp3, cp4: integer;
begin
  checksym(loop1,133);
  if cp /= -1 then
    cp0 := cp;
  else
    cp0 := -1;
  end if;
  if cpw = -1 then
     cp4 := codept + 1;
  else
     cp4 := cpw;
  end if;
  lstat_seq(lvl, plvl, cp0);
  coder1(jmp3, cp4);
  cp3 := codept + 1;
  cp1 := cp0;
-- fix - up for exit statements in a loop
  while cp1 /= -1 loop
    cp2 := cp1;
```

```
    cp1 := code(cp1).opnd1;
    code(cp2).opnd1 := cp3;
  end loop;
  checksym(end1,134);
  checksym(loop1,135);
end;

procedure while_stat(lvl, plvl: integer) is
tpt, cp1: integer;
begin
  checksym(while1,137);
  cp1 := codept + 1;
  expression(lvl, plvl, tpt);
  if tpt /= bool_pt then
    debug.error(138, symbols'pos(bool1));
  end if;
  coder1(jz3, -1);
  loop_stat(lvl, plvl, codept, cp1);
end;

procedure comp_stat(lvl, plvl: integer) is
begin
   case sym is
    when if1 => if_stat(lvl, plvl);
    when loop1 => loop_stat(lvl, plvl, -1, -1);
    when while1 => while_stat(lvl, plvl);
    when others => debug.error(106, symbols'pos(id1));
   end case;
end;

procedure statement(lvl, plvl: integer) is
begin
   case sym is
    when id1 | null1 | write1 | writeln1 | read1 =>
```

```
simp_stat(lvl, plvl); checksemi(200);
    when if1 | loop1 | while1 => comp_stat(lvl, plvl); checksemi(201);
    when exit1 => debug.error(400, symbols'pos(exit1));
    when others => debug.error(90, symbols'pos(if1));
  end case;
end;

procedure lstatement(lvl, plvl: integer; cp: in out integer) is
begin
  case sym is
    when id1 | null1 | write1 | writeln1 | read1 |
 if1 | loop1 | while1 => statement(lvl, plvl);
    when exit1 => exit_stat(lvl, plvl, cp); checksemi(401);
    when others => debug.error(402, symbols'pos(if1));
  end case;
end;

procedure stat_seq_tail(lvl, plvl: integer) is
begin
  case sym is
   when id1 | null1 | if1 | loop1 | while1 |
write1 | writeln1 | read1 =>
       statement(lvl, plvl);
       stat_seq_tail(lvl, plvl);
   when end1 | elsif1 | else1 => null;
   when exit1 => debug.error(403, symbols'pos(exit1));
   when others =>
      debug.error(85, symbols'pos(id1));
  end case;
end;

procedure lstat_seq_tail(lvl, plvl: integer; cp: in out integer) is
begin
  case sym is
```

```
   when id1 | null1 | if1 | exit1 | loop1 | while1 |
write1 | writeln1 | read1 =>
      lstatement(lvl, plvl, cp);
      lstat_seq_tail(lvl, plvl, cp);
   when end1  => null;
   when others =>
     debug.error(411, symbols'pos(id1));
  end case;
end;

procedure stat_seq(lvl, plvl: integer) is
begin
  statement(lvl, plvl);
  stat_seq_tail(lvl, plvl);
end;

procedure lstat_seq(lvl, plvl: integer; cp: in out integer) is
begin
  lstatement(lvl, plvl, cp);
  lstat_seq_tail(lvl, plvl, cp);
end;

procedure decl_part(lvl, plvl:integer; displ: in out integer);

procedure subprog_part(lvl, plvl, displ, tp: integer) is
disp, pt, sz1, sz2: integer;
begin
   disp := displ;
   pkg_d_pt := pkg_d_pt + 1;
   pkg_d_stack(pkg_d_pt) := 0;
   decl_part(lvl, plvl, disp);
   if pkg_d_stack(pkg_d_pt) /= 0 then
     debug.error(112, symbols'pos(pack1));
   else
```

```
    pkg_d_pt := pkg_d_pt - 1;
   end if;
   checksym(begin1,50);
   symbol_table(tp).size := codept + 1;
   symbol_table(tp).lo := 1;
   sz1 := 0; sz2 := 0; pt := symbol_table(tp).sect_end;
   while pt > tp loop
    if symbol_table(pt).kind = obj4 or
symbol_table(pt).kind = param4 then
sz1 := sz1 + symbol_table(pt).size;
    end if;
    if symbol_table(pt).kind = param4 then
sz2 := sz2 + symbol_table(pt).size;
    end if;
pt := symbol_table(pt).bkwrd;
   end loop;
   symbol_table(tp).dsp := sz1 + dispconst;
   symbol_table(tp).pdsp := sz2 + dispconst;
   stat_seq(lvl, plvl);
   coder1(ret3, lvl);
   checksym(end1, 51);
end;

procedure subprog_decl(lvl, plvl: integer) is
tp, disp: integer;
begin
  subprog_header(lvl,plvl,disp,tp);
  checksym(is1, 30);
  subprog_part(lvl + 1,plvl+1,disp,tp);
  checksemi(31);
end;

procedure pkg_d_decl(lvl, plvl: integer; displ: in out integer) is
dummy: integer;
```

```
begin
   case sym is
    when id1 => object_decl(lvl, plvl, displ); pkg_d_decl(lvl, plvl, displ);
    when type1 => type_decl(lvl, plvl); pkg_d_decl(lvl, plvl, displ);
    when proc1 => subprog_header(lvl, plvl, displ,dummy); checksemi(64);
   pkg_d_decl(lvl, plvl, displ);
    when end1 => null;
    when others => debug.error(75, symbols'pos(pack1));
   end case;
end;

procedure pkg_def_decl(lvl, plvl: integer; displ: in out integer) is
begin
 pkg_d_stack(pkg_d_pt) := pkg_d_stack(pkg_d_pt) + 1;
 pkg_d_decl(lvl, plvl, displ);
 checksym(end1,61);
 checksym(pack1,62);
 checksemi(63);
end;

procedure pkg_b_decl(lvl, plvl:integer; displ: in out integer) is
begin
   case sym is
    when id1 => object_decl(lvl, plvl, displ); pkg_b_decl(lvl, plvl, displ);
    when type1 => type_decl(lvl, plvl); pkg_b_decl(lvl, plvl, displ);
    when proc1 => subprog_decl(lvl, plvl);
   pkg_b_decl(lvl, plvl, displ);
    when end1 => null;
    when others => debug.error(74, symbols'pos(pack1));
   end case;
end;

procedure pkg_body_decl(lvl, plvl: integer; displ: in out integer) is
begin
```

```
  pkg_d_stack(pkg_d_pt) := pkg_d_stack(pkg_d_pt) - 1;
  pkg_b_decl(lvl, plvl, displ);
  checksym(end1,70);
  checksym(pack1,71);
  checksemi(72);
end;

procedure package_decl(lvl, plvl: integer; displ: in out integer);

function proc_check(psn1, psn2: integer) return boolean is
begin
  return true;
end;

procedure fix_p_addr(adr, disp, pt: integer) is
cpt, cpt2: integer;
begin
   cpt := symbol_table(pt).size;
   if symbol_table(pt).lo = -1 then
      symbol_table(pt).size := adr;
      symbol_table(pt).dsp := disp;
      symbol_table(pt).lo := 1;
      while cpt /= -1 loop
cpt2 := cpt;
cpt := code(cpt).opnd2;
code(cpt2).opnd2 := adr;
code(cpt2).opnd3 := disp;
      end loop;
   end if;
end;

procedure pkg_search(pp,tp: integer) is
pt1, pt2, pt3, nm, temp: integer; ok: boolean;
addr, fix_addr: integer;
```

```
begin
-- pp is the pointer into the def
-- tp is the pointer into the body
   pt1 := symbol_table(pp).sect_end;
   pt2 := symbol_table(tp).sect_end;
   temp := symbol_table(tp).bkwrd;
   symbol_table(tp).bkwrd := 0;
   while pt1 /= pp loop
     if symbol_table(pt1).kind = proc_id4 then
nm := symbol_table(pt1).id_name;
pt3 := pt2;
symbol_table(0).id_name := nm;
while symbol_table(pt3).id_name /= nm loop
  pt3 := symbol_table(pt3).bkwrd;
end loop;
fix_p_addr(symbol_table(pt3).size,symbol_table(pt3).dsp,pt1);
     end if;
     pt1 := symbol_table(pt1).bkwrd;
   end loop;
   symbol_table(tp).bkwrd := temp;
end;

procedure pkg_tail(lvl, plvl: integer; displ: in out integer) is
pl, j, tp, pp: integer;
begin
    case sym is
     when id1 =>
       check_new_id(2, secondsym, lvl, pl);
 if pl = 0 then
    enter_var(secondsym, lvl, plvl, pkg_id4);
    symbol_table(table_pt).typ := pkg_def3;
          display(lvl+1) := table_pt;
          symbol_table(table_pt).sect_end := table_pt;
    symbol_table(table_pt).pk_pt := table_pt;
```

```
 else debug.error(53,symbols'pos(pack1)); end if;
       nextsym;
       checksym(is1,54);
 pkg_def_decl(lvl+1, plvl, displ);
     when body1 =>
       nextsym;
       if sym = id1 then
         check_new_id(2, secondsym, lvl, pl);
         if pl > 0 then
           if symbol_table(pl).typ /= pkg_def3 then
             debug.error(59, symbols'pos(pack1));
           else
enter_var(secondsym, lvl, plvl, pkg_id4);
             tp := table_pt;
      symbol_table(table_pt).typ := pkg_bdy3;
             symbol_table(table_pt).pk_pt := pl;
             pp := pl;
             display(lvl+1) := table_pt;
             symbol_table(table_pt).sect_end := table_pt;
           end if;
         else
           debug.error(60,symbols'pos(pack1));
         end if;
         nextsym;
        else
         debug.error(61, symbols'pos(id1));
        end if;
        checksym(is1,56);
  pkg_body_decl(lvl+1, plvl, displ);
        pkg_search(pp, tp);
       --end when body1
      when others => debug.error(61,symbols'pos(pack1));
   end case;
end;
```

```
procedure package_decl(lvl, plvl: integer; displ: in out integer) is
begin
  checksym(pack1, 52);
  pkg_tail(lvl, plvl, displ);
end;

procedure declaration(lvl, plvl:integer; disp: in out integer) is
begin
  case sym is
   when id1 => object_decl(lvl, plvl, disp);
   when type1 => type_decl(lvl, plvl);
   when proc1 => subprog_decl(lvl, plvl);
   when pack1 => package_decl(lvl, plvl, disp);
   when others => debug.error(2, symbols'pos(sym));
  end case;
end;

procedure decl_part(lvl, plvl:integer; displ: in out integer) is
disp: integer;
begin
    disp := displ;
    if sym = type1 or sym = id1 or sym = proc1 or sym = pack1 then
      declaration(lvl, plvl, disp);
      decl_part(lvl, plvl, disp);
    elsif sym /= begin1 then
      debug.error(52, symbols'pos(proc1));
    end if;
    displ := disp;
end;

procedure program is
begin
   subprog_decl(-1, -1);
```

```
end;

procedure emit_code(n: integer) is
use int_io;
sz, sz0, sz1, sz2, sz3, opval: integer;
op: operation;
begin
  sz := 0;
  for i in 1 .. n loop
    opval := code(i).oprtn;
    op := operation'val(opval);
    sz0 := debug.size(opval);
    put(codefi,opval,sz0);
    text_io.put(codefi,' ');
    sz := sz + sz0 + 1;
    if sz >= 64 then
      text_io.new_line(codefi);
      sz := 0;
    end if;
    if op_sing(op) then
      sz1 := debug.size(code(i).opnd1);
      put(codefi, code(i).opnd1,sz1);
      text_io.put(codefi,' ');
      sz := sz + sz1 + 1;
      if sz >= 64 then
text_io.new_line(codefi);
sz := 0;
      end if;
    elsif op_doub(op) then
      sz1 := debug.size(code(i).opnd1);
      sz2 := debug.size(code(i).opnd2);
      put(codefi, code(i).opnd1,sz1);
      text_io.put(codefi,' ');
      put(codefi, code(i).opnd2,sz2);
```

```
        text_io.put(codefi,' ');
        sz := sz + sz1 + sz2 + 2;
        if sz >= 64 then
  text_io.new_line(codefi);
  sz := 0;
        end if;
      elsif op_trip(op) then
        sz1 := debug.size(code(i).opnd1);
        sz2 := debug.size(code(i).opnd2);
        sz3 := debug.size(code(i).opnd3);
        put(codefi, code(i).opnd1,sz1);
        text_io.put(codefi,' ');
        put(codefi, code(i).opnd2,sz2);
        text_io.put(codefi,' ');
        put(codefi, code(i).opnd3,sz3);
        text_io.put(codefi,' ');
        sz := sz + sz1 + sz2 + sz3 + 3;
        if sz >= 64 then
  text_io.new_line(codefi);
  sz := 0;
        end if;
      end if;
    end loop;
    text_io.close(codefi);
end;

begin
    getfile;
    nextsym;
    coder1(mbl3,0);
    coder3(cal3,0,-1,-1);
    coder0(hlt3);
    program;
    code(2).opnd1 := symbol_table(5).lvl+1;
```

```
    code(2).opnd2 := symbol_table(5).size;
    code(2).opnd3 := symbol_table(5).dsp;
    coder0(endm3);
--  show_ops;
--  show_table2;
    emit_code(codept);
    exception
      when text_io.name_error =>
        text_io.put_line("File 'out1' not found");
      when err_excep =>
        text_io.put_line("Error");
      when coderr =>
        text_io.put_line("Too much code");
end pass3;
```

Pass4: The a-machine.

The a-machine is a stack machine patterned after the Pascal p-machine. This machine interprets the code produced by pass3. This kind of machine was used in an early version of Pascal. It can be found in [W] and also in [E]. It presents an excellent alternative to producing machine code or assembly code because it is written in a high level language and is more easily understood. Eventually, for use in a working compiler it can be translated into machine code and optionally an optimization pass can be placed prior to it. We list the a-machine here.

```
with text_io;
package int_io is new text_io.integer_io(integer);
with text_io; with int_io;

procedure pass4 is
--   operations
  type operation is (add3, and3, bld3, blmd3, blmi3, cal3, div3, --6
    endm3, eq3, --8
    ge3, gt3, hlt3, jmp3, jnz3, jz3, lad3,--15
    lai3, le3, lit3, lod3, lt3, mbl3, mod3,--23
    mul3, ne3, neg3, nop3, not3, or3, rdb3, rdi3, --31
    ret3, sto3, sub3, wrtb3, wrti3); --36
--   end operations
   op_sing, op_doub, op_trip: array(operation) of boolean :=
 (add3 .. wrti3 => false);
   type order is record
   oprtn: operation;
   opnd1, opnd2, opnd3: integer;
end record;
   codelim: constant:= 500;
   code: array(1 .. codelim) of order;
   codefi, outx: text_io.file_type;
   display: array(integer range -1 .. 50) of integer;
```

```
   stack: array(integer range 1 .. 500) of integer;
   stkpt : integer;
   ip: integer; stop: boolean;
   subtype str7 is string(1 .. 7);
   type oprek is record
   opnm:str7;
   oplen:integer;
end record;
   opmnem: array(operation) of oprek;
   last_code: integer;

procedure init is
code_index: integer; op: operation; f_op, opd: integer;
use text_io;
begin
    open(codefi,in_file, "code");
    create(outx, out_file, "outp");
    op_sing(lit3) := true;
    op_sing(mbl3) := true;
    op_sing(jmp3) := true;
    op_sing(jnz3) := true;
    op_sing(jz3) := true;
    op_sing(wrti3) := true;
    op_sing(wrtb3) := true;
    op_trip(cal3) := true;
    op_doub(lad3) := true;
    op_doub(lai3) := true;
    op_sing(bld3) := true;
    op_doub(blmd3) := true;
    op_doub(blmi3) := true;
    op_sing(ret3) := true;
  opmnem  := (("add3   ", 4),
      ("and3   ", 4),
      ("bld3   ", 4),
```

```
        ("blmd3  ", 5),
        ("blmi3  ", 5),
        ("cal3   ", 4),
        ("div3   ", 4),
        ("endm3  ", 5),
        ("eq3    ", 3),
        ("ge3    ", 3),
        ("gt3    ", 3),
        ("hlt3   ", 4),
        ("jmp3   ", 4),
        ("jnz3   ", 4),
        ("jz3    ", 3),
        ("lad3   ", 4),
        ("lai3   ", 4),
        ("le3    ", 3),
        ("lit3   ", 4),
        ("lod3   ", 4),
        ("lt3    ", 3),
        ("mbl3   ", 4),
        ("mod3   ", 5),
        ("mul3   ", 4),
        ("ne3    ", 3),
        ("neg3   ", 4),
        ("nop3   ", 4),
        ("not3   ", 4),
        ("or3    ", 3),
        ("rdb3   ", 4),
        ("rdi3   ", 4),
        ("ret3   ", 4),
        ("sto3   ", 4),
        ("sub3   ", 4),
        ("wrtb3  ", 5),
        ("wrti3  ", 5));
    code_index:= 0;
```

```
 loop
    int_io.get(codefi, f_op);
    op := operation'val(f_op);
  code_index := code_index + 1;
  code(code_index).oprtn := op;
  if op_sing(op) then
    int_io.get(codefi, code(code_index).opnd1);
  elsif op_doub(op) then
    int_io.get(codefi, code(code_index).opnd1);
    int_io.get(codefi, code(code_index).opnd2);
  elsif op_trip(op) then
    int_io.get(codefi, code(code_index).opnd1);
    int_io.get(codefi, code(code_index).opnd2);
    int_io.get(codefi, code(code_index).opnd3);
  end if;
 exit when op = endm3;
  end loop;
  last_code := code_index;
  stop := false;
end;

procedure execute is
  op: operation;
  opd1, opd2, opd3, hold, base: integer;
  t1, t2, num: integer;
  procedure show_stack is
  begin
  text_io.put("stkpt =");int_io.put(stkpt);text_io.new_line;
  text_io.put("base =");int_io.put(base);text_io.new_line;
  for i in 1 .. stkpt loop
    int_io.put(i,4); text_io.put(":  "); int_io.put(stack(i));
    text_io.new_line;
  end loop;
  end;
```

```
begin
-- base points to the beginning of an activation record
-- base + 1 is the dynamic pointer
-- base + 2 is the static pointer
-- static pointers are maintained by means of a display
  ip := 1;
  base := -1;
  stkpt := 1;
  display(0) := -1;
  loop
    op := code(ip).oprtn;
    if op_sing(op) then
      opd1 := code(ip).opnd1;
    elsif op_doub(op) then
      opd1 := code(ip).opnd1;
      opd2 := code(ip).opnd2;
    elsif op_trip(op) then
      opd1 := code(ip).opnd1;
      opd2 := code(ip).opnd2;
      opd3 := code(ip).opnd3;
    end if;
    ip := ip + 1;
    case op is
     when add3 =>
      stkpt := stkpt - 1;
      stack(stkpt) := stack(stkpt) + stack(stkpt + 1);
     when and3 =>
      stkpt := stkpt - 1;
      if stack(stkpt) /= 0 then
 stack(stkpt) := stack(stkpt + 1);
      end if;
     when bld3 =>
      t2 := stack(stkpt);
```

```
        t1 := stack(stkpt - 1);
        for i in 0 .. opd1 - 1 loop
stack(t1 + i) := stack(t2 + i);
        end loop;
        stkpt := stkpt - 2;
       when blmi3 =>
        t1 := stack(stkpt);
        t2 := base + opd1;
        for i in 0 .. opd2 - 1 loop
stack(t2 + i) := stack(t1 + i);
        end loop;
        stkpt := stkpt - 1;
       when blmd3 =>
        stack(base + opd1) := stack(stkpt);
        stkpt := stkpt - 1;
       when cal3 =>
        stkpt := base + opd3 - 1;
        stack(base):= ip;
        stack(base+2) := display(opd1);
        display(opd1) := base;
        ip := opd2;
       when div3 =>
        stkpt := stkpt - 1;
        stack(stkpt) := stack(stkpt) / stack(stkpt + 1);
       when eq3 =>
        stkpt := stkpt - 1;
        stack(stkpt) := boolean'pos(stack(stkpt) = stack(stkpt+1));
       when ge3 =>
        stkpt := stkpt - 1;
        stack(stkpt) := boolean'pos(stack(stkpt) >= stack(stkpt+1));
       when gt3 =>
        stkpt := stkpt - 1;
        stack(stkpt) := boolean'pos(stack(stkpt) > stack(stkpt+1));
       when hlt3 => stop := true;
```

```
    when jmp3 => ip := opd1;
    when jnz3 =>
     if stack(stkpt) /= 0 then
ip := opd1;
     end if;
     stkpt := stkpt - 1;
    when jz3 =>
     if stack(stkpt) = 0 then
ip := opd1;
     end if;
     stkpt := stkpt - 1;
    when lad3 =>
 stkpt := stkpt + 1;
 stack(stkpt) := display(opd1) + opd2;
    when le3 =>
     stkpt := stkpt - 1;
     stack(stkpt) := boolean'pos(stack(stkpt) <= stack(stkpt+1));
    when lit3 =>
     stkpt := stkpt + 1;
     stack(stkpt) := opd1;
    when lai3 =>
 stkpt := stkpt + 1;
 stack(stkpt) := stack(display(opd1) + opd2);
    when lod3 =>
     stack(stkpt) := stack(stack(stkpt));
    when lt3 =>
     stkpt := stkpt - 1;
     stack(stkpt) := boolean'pos(stack(stkpt) < stack(stkpt+1));
    when mbl3 =>
     stack(stkpt+2) := base;
     base := stkpt + 1;
     stkpt := base + opd1;
    when mod3 =>
     stkpt := stkpt - 1;
```

```
        stack(stkpt) := stack(stkpt)  mod  stack(stkpt + 1);
       when mul3 =>
        stkpt := stkpt - 1;
        stack(stkpt) := stack(stkpt) * stack(stkpt + 1);
       when ne3 =>
        stkpt := stkpt - 1;
        stack(stkpt) := boolean'pos(stack(stkpt) /= stack(stkpt+1));
       when neg3 => stack(stkpt) := - stack(stkpt);
       when not3 => stack(stkpt) := 1 - stack(stkpt);
       when or3 =>
        stkpt := stkpt - 1;
        if stack(stkpt) = 0 then
  stack(stkpt) := stack(stkpt + 1);
        end if;
       when rdb3 =>
        int_io.get(hold);
        if (hold /= 0) and (hold /= 1) then
   text_io.put_line(outx, "Boolean read error ");
   stop := true;
       else
   stkpt := stkpt + 1;
   stack(stkpt) := hold;
        end if;
      when rdi3 =>
        text_io.put("?");
        stkpt := stkpt + 1;
        int_io.get(stack(stkpt));
      when ret3 =>
        display(opd1) := stack(base + 2);
        ip := stack(base);
        stkpt := base - 1;
        base := stack(base + 1);
      when sto3 =>
        stack(stack(stkpt - 1)) := stack(stkpt);
```

```
      stkpt := stkpt - 2;
     when sub3 =>
      stkpt := stkpt - 1;
      stack(stkpt) := stack(stkpt) - stack(stkpt + 1);
     when wrtb3 =>
      if stack(stkpt) = 0 then
 text_io.put_line(outx,"false");
      elsif stack(stkpt) = 1 then
 text_io.put_line(outx, "true");
      end if;
      stkpt := stkpt - 1;
     when wrt:3 =>
      t1 := stack(stkpt);
      stkpt := stkpt - 1;
      if t1 > 0 then
 int_io.put(outx, stack(stkpt), t1);
      else
 int_io.put(outx, stack(stkpt), 10);
      end if;
      text_io.new_line(outx);
      stkpt := stkpt - 1;
      when others => null;
    end case;
    exit when stop;
  end loop;
end;


procedure show_ops is
op: operation;
begin
 for i in 1 .. last_code loop
   int_io.put(i,4);text_io.put(":  ");
   op := code(i).oprtn;
```

```
    text_io.put(opmnem(op).opnm);
    if op_sing(op) then
      int_io.put(code(i).opnd1,6);
    elsif op_doub(op) then
      int_io.put(code(i).opnd1,6);
      int_io.put(code(i).opnd2,6);
    elsif op_trip(op) then
      int_io.put(code(i).opnd1,6);
      int_io.put(code(i).opnd2,6);
      int_io.put(code(i).opnd3,6);
    end if;
    text_io.new_line;
  end loop;
  text_io.put("last_code = ");
  int_io.put(last_code);
  text_io.new_line;
end;

begin
  init;
--  show_ops;
  execute;
  text_io.close(outx);
end pass4;
```

Conclusion:

We believe that the course we produced was quite successful. Ada supported the design and programming features of our example compiler quite well. If Ada can be faulted it probably lies with the somewhat large storage overhead in both files and memory. On the other hand we found that the Ada syntax and the strong typing supported good program design. We believe these features o Ada caught many problems before they became large problems. The programming proceeded with a minimum of difficulty. The experience has shown that Ada can be used with great utility for a compiler design course. The writer plans to continue to use Ada in this course in the future.

The course compiler in machine readable form is available in a disk attached to the cover of the report. Please read the file called READ.ME first.